A CRAY RESEARCH, INC. PUBLICATION

CRAY CHANNELS

Volume 3, Number 1

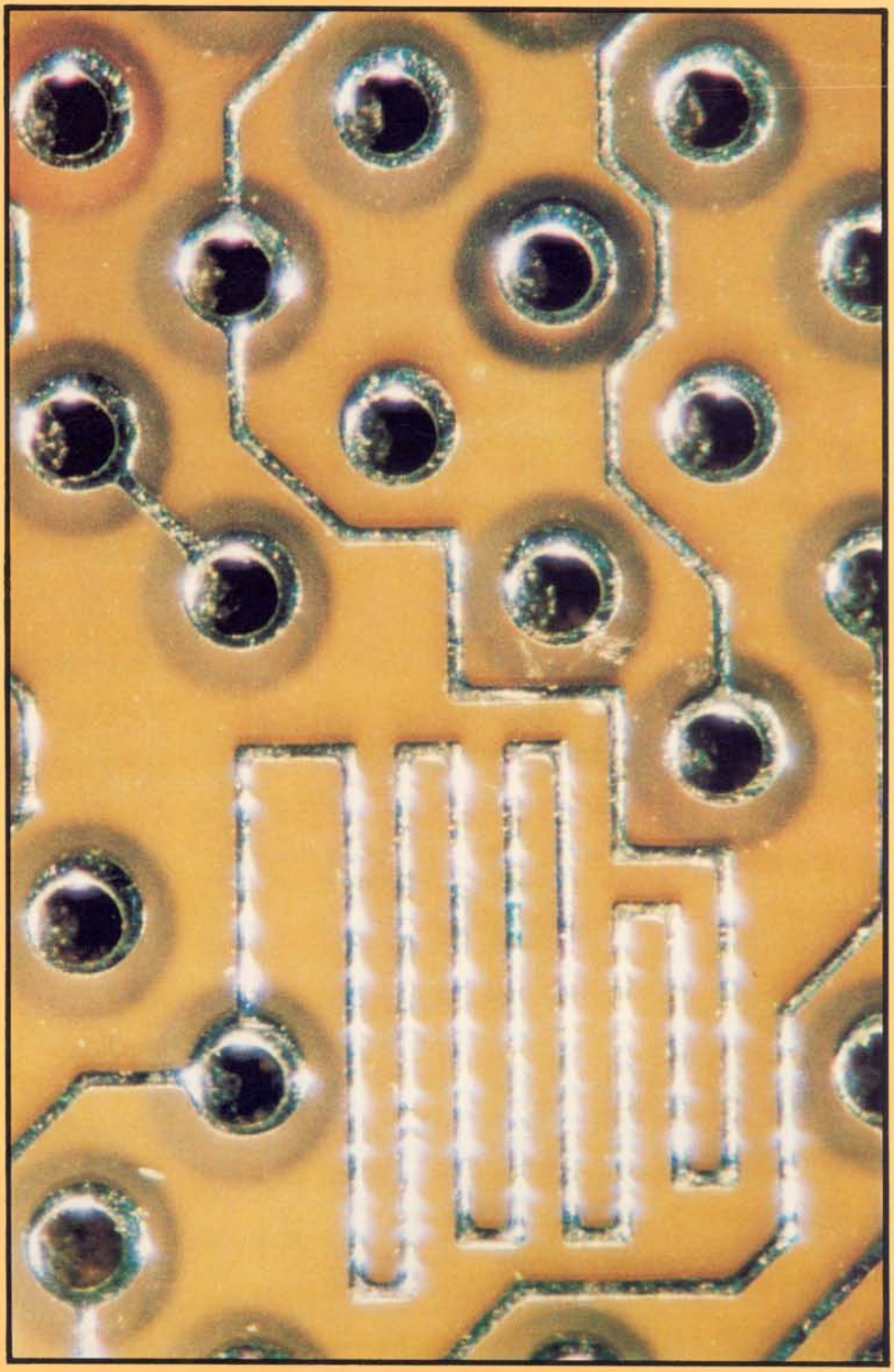
FEATURE ARTICLES:

Direct solution of linear equations on the CRAY-1

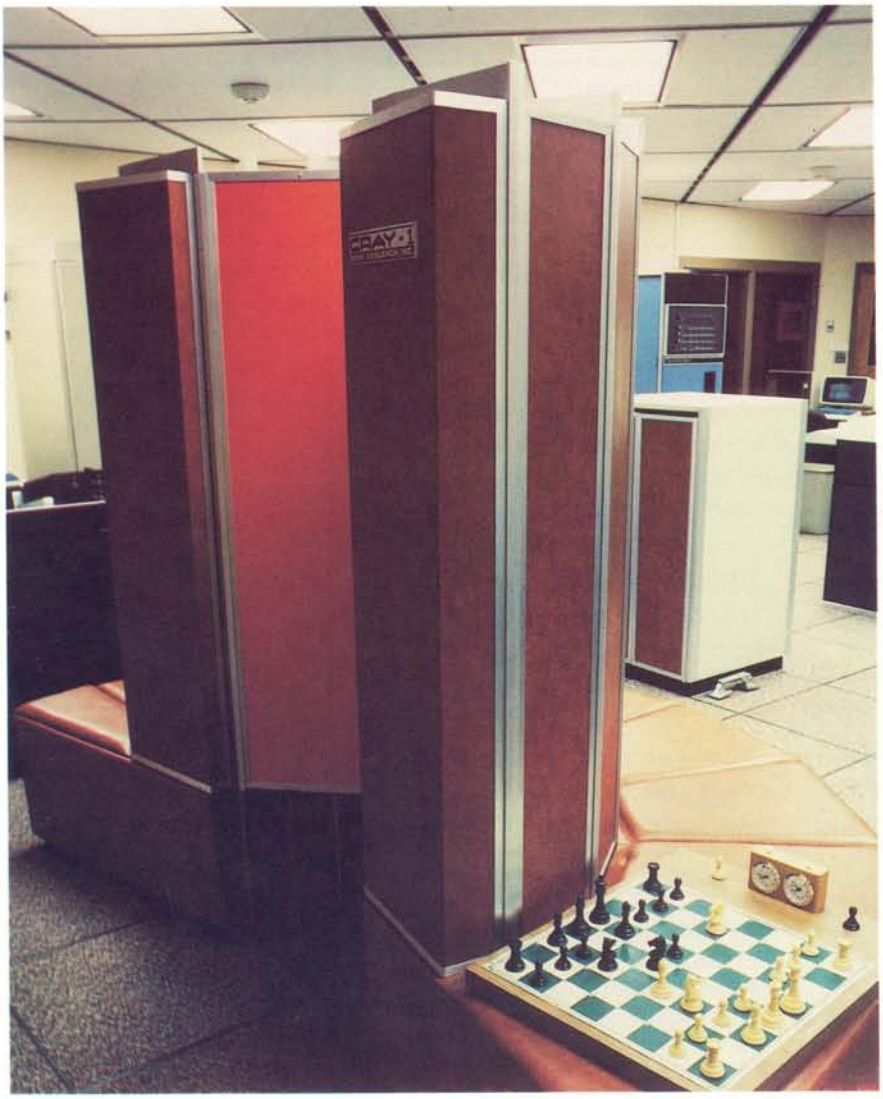
CHECKMATE! — The CRAY-1 plays chess

AND OUR REGULAR COLUMNS:

- Applications in depth
- Scientific applications package highlight
- Corporate register
- Software release summary
- and user news



CHECKMATE — T



For the past year, Cray Research has provided computer time for computer chess program development and tournament action. The rigors of chess provide a strenuous trial for both hardware and software, giving the company an additional testing ground. The fact that CRAY BLITZ happens to be one of the top-ranked computer chess programs in the world is just frosting on the cake.

he CRAY-1 plays chess

Robert Hyatt
University of Southern Mississippi, Hattiesburg

Part one of a two-part series

"Can computers think?" This has been one of the most controversial questions provoked by the modern digital computer. It has been proven that for finite mathematics where exact solutions exist, the computer is vastly superior to man in finding the solution. However, man has always had a clear superiority in inexact problems, due to such attributes as intuition, hunches and other non-quantifiable decision-making processes. The computer is rapidly eliminating these advantages as hardware speeds improve and programming tools are redesigned.

Since chess has long been considered an "intellectual" exercise, it was natural that the computer would be applied to its solution. While there is still a significant difference in playing strength between world-champion class players and the computer, this difference is shrinking yearly. In fact, most computer scientists believe that it is only a matter of time before computers become unbeatable. As evidence of this, six years ago chess masters easily defeated computers in speed chess, simultaneous exhibitions and regular tournament chess. In a period of six years, the top programs have sharply reduced this gap. Top human players struggle to win speed chess games and simultaneous exhibition games and even world-champion class players have fallen victim to computers in speed chess. Chess masters are beginning to find that they lose regularly to the better programs, even in tournament-level chess matches. If chess programs continue to improve over the next decade as they have over the past decade, the programs could easily invade the ranks of the top 50 players in the world and have a significant chance at becoming world champions.

Most computer scientists believe that it is only a matter of time before computers become unbeatable.

INTERNAL BOARD REPRESENTATION

The logical chess board is an 8 by 8 array representing the 64 board squares. To allow rapid detection of the edge of the board during move generations, a double row of border squares is used (Figure 1). Squares outlined by the dark lines are on the board and squares outside of the dark lines are illegal squares.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Figure 1. *Logical chess board*

Pieces are represented as digits 1 through 6, with pawn = 1, knight = 2, bishop = 3, rook = 4, queen = 5, and king = 6. An empty square has a value of 0, and a square off of the board has a value of 99. To distinguish program pieces from the opponent's pieces, program pieces are given positive numbers and the opponent's pieces are negative numbers.

Move generation with this representation is simple. For example, let's generate the moves of a program rook on square 22. The four legal directions for a rook are +1, -1, +10, and -10. The rook can move in the +1 direction to squares 23, 24, 25, 26, 27, 28, or 29 unless one of them is occupied. If a square is occupied (a number other than 0), the rook cannot move beyond it. Also, if the sign of the occupying piece is +, the rook cannot capture it. These steps are repeated for the other three directions to enumerate all legal rook moves. Note that for the -1 and -10

directions, a 99 will immediately terminate generation of that direction, indicating that the edge of the board has been passed.

Because, during the course of a game, each side has an average of 38 legal moves, a tree of depth 6 has 38^6 or 3,010,936,384 terminal positions!

GAME TREE SEARCH

The game tree search is the tactical component of the move selection process. The tree search will find (within reason) move sequences that will avoid losing material whenever material loss is threatened, and it will find move sequences that will win material whenever the opportunity arises.

The basic tree search is sometimes called a minimax full-width depth-first game tree search. The program is given a position for which it must try to find the best move. This position is referred to as the "root" or "base" of the tree. From this position, the program generates all of the possible moves and stores them in an array. The first move is selected and made on the internal game board. Now, the program switches sides and generates all possible moves from this new position. The first move is again selected and made, and this process continues until some fixed depth is reached.

At this point, the evaluation function is used to compute a numeric value representing the current position. The larger the value, the better for the program; the smaller the value, the better for the opponent. The side to move stores this score and selects the next move. After all moves at this level have been examined, the side to move will have chosen the score that was highest or lowest, depending on which side is to move. This score is returned to the previous level, where it is stored, and a new move is selected. Once again the program follows the process of advancing to the next level, generating all moves and making one, until the maximum depth is reached. As all moves for a level are examined, the best (worst) score is returned to the previous level. This continues until all moves at the base level are examined.

As can be seen, the game tree is quite large. Because, during the course of a game, each side has an average of 38 legal moves, a tree of depth 6 has 38^6 or 3,010,936,384 terminal positions! Even if the program is only using one microsecond of processor time per node, this is still nearly one hour of computation per move.

To speed up the process, then, there is a technique known as the alpha-beta backward pruning algorithm that drastically reduces the number of positions evaluated without affecting the ultimate outcome of the search. Suppose that the first move the program examines is N-B7 check (Figure 2). The tree search concludes that this wins a rook and returns a value of +5. When the program selects the next move to analyze, the alpha-beta algorithm may help as follows: after changing sides, generating all moves and selecting one, and doing the additional searching necessary to evaluate the resulting position, the program discovers that after it plays P-KR3, the opponent plays P-QR3, and it does not win a rook. The program could continue analyzing moves at level two and might even find that P-KR3, B-B4 wins a pawn. However, this is wasted time for two reasons: 1) N-B7+ wins a rook, and 2) P-KR3 wins nothing and may even lose something. Therefore, the program ignores P-KR3 and selects another move.

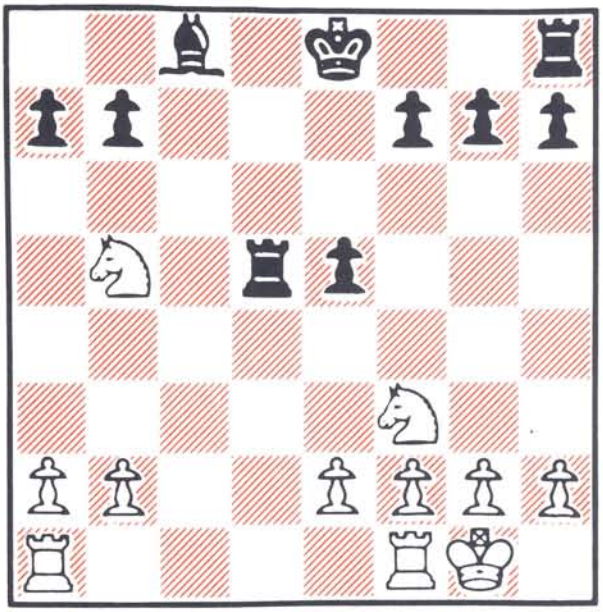


Figure 2. *White to move*

The above strategy is analogous to the following example: suppose you have on the table ten bowls of stew, and you must select one to eat. Some are far too hot, some are too cold, some are spoiled, and some are improperly cooked. Suppose the first bowl you select is slightly cool, but otherwise is acceptable. Examination of the next bowl finds it boiling hot. Testing it to determine if it tastes right is futile, since it can be rejected based on being hotter than something already found.

Using this algorithm can potentially reduce the number of nodes from 38^6 to $2x38^3$ or 109,744. In practice, this is difficult to achieve, but dramatic savings are possible nonetheless. On the CRAY-1, for example, a typical six ply search might examine 200,000 nodes, which is far less than the potential maximum.

There are many other useful algorithms that can help speed up the search. First, the alpha-beta algorithm depends on move ordering being

quite accurate. For example, in the situation mentioned above, if the program had examined P-KR3 first, then it would also have to completely examine N-B7+ to evaluate the move, because alpha-beta would not terminate analysis early.

In Figure 3, the program must defend against the move N-B7+. However, since the tree search is responsible for tactics, the program will simply generate all moves at level one and make the first one. After changing sides and analyzing all of the replies, the program determines that the first move, P-K3, is answered by N-B7+ where the opponent wins a rook. This move, N-B7+ is remembered as a "killer" move for level two. Now, after trying P-K4, the killer move is tried first. This gives good moves a chance to be examined first, causing more alpha-beta cutoffs. In fact, until R-QB1 is examined, all of the program's moves are refuted by N-B7+.

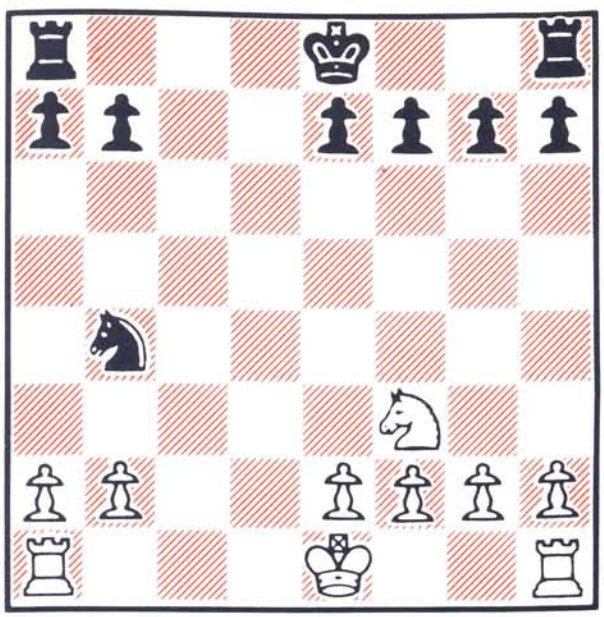


Figure 3. White to move

CRAY BLITZ maintains (remembers) up to ten killers per level in the tree. The list is carefully ordered so that the more useful killers are examined first.

Before continuing, it is worth mentioning that the killer heuristic, as well as others that improve the tree search, does not alter the move that the program ultimately will make. However, it does alter drastically the amount of time it will take to perform the search. In fact, removing the killer logic completely will increase the search time by a factor of at least ten. This is the principal advantage of the full-width or exhaustive search — that move-ordering can affect the speed of the search, but not the outcome.

Another time-saving algorithm used in CRAY BLITZ is the transposition table. When doing a six ply tree search, many different pathways may converge to the same position. For example, the move sequences N-KB3, B-QB4, N-KN5 and N-KR3, B-QB4, N-KN5 both converge to the same position. After reaching this position from one branch of the tree, and then doing an additional three ply search from this point, the result of this search is saved in the transposition table. After playing N-KR3, B-QB4, N-KN5, this position is found in the table along with the search value. This avoids analyzing the subtree that would normally be developed. This algorithm results in a speed improvement of 2 to 5 in the middle game and a speed improvement of over 100 in simple endgames.

The killer heuristic does not alter the move that the program ultimately will make. However, it does alter drastically the amount of time it will take to perform the search.

QUIESCENCE

Quiescence is the single most important consideration in a computer chess program. The problem centers around when to apply the positional evaluation, or more technically, raises the question of when a position is tactically stable enough to be accurately evaluated. For example, when performing a three ply search, the move sequence R-K1, R-K1, RXR seems to win a rook; however, suppose the next move (if the program were searching to four ply) is NXQ. If the positional evaluator were applied after these moves, it would conclude that the program is winning a rook (remember that tactics are handled by the search only).

Since terminating the search at some arbitrary fixed depth results in many blunders, most chess programs carry out the tree search in three distinct phases. Phase One is the full-width or exhaustive part of the search, and is carried out to some predetermined depth based on timing requirements. Phase Two is the quiescence part of the search and is carried out until the quiescence search encounters a position that is tactically stable. Phase Three is the positional evaluator, which can now be applied because no pieces are hanging or no threats exist.

The quiescence search differs from the full-width search in that only a small subset of the legal moves are examined — notably, captures to stabilize the material balance and checks to find/avoid checkmates. Unfortunately, this area is key to why a computer is not the current world champion. It is extremely difficult for a chess program to determine if a position is tactically quiescent.

For example, if the position in Figure 4 were encountered during the quiescence search, it might be judged as a terminal position, since no material is hanging. However, white can attack the black knight, which is pinned, more times that black can defend it, winning at least a piece.

While a human easily sees this and avoids it, a computer has a great deal of difficulty with it. CRAY BLITZ examines winning captures and checks in the quiescence search, as do most other top-flight programs, and all would improperly evaluate this position if it occurs in the quiescence search.

Another problem of quiescence is known as the "horizon effect". Simply, if a program can delay something so that it is not discerned within the search, it does not exist! In Figure 4, if black continually attacks white pieces with its pawns and pieces, white doesn't have time to continue attacking the knight to win it. A human understands that delaying the attacks does not eliminate them, but the computer, by forcing the attacks out of the search, thinks that it has totally avoided the problem.

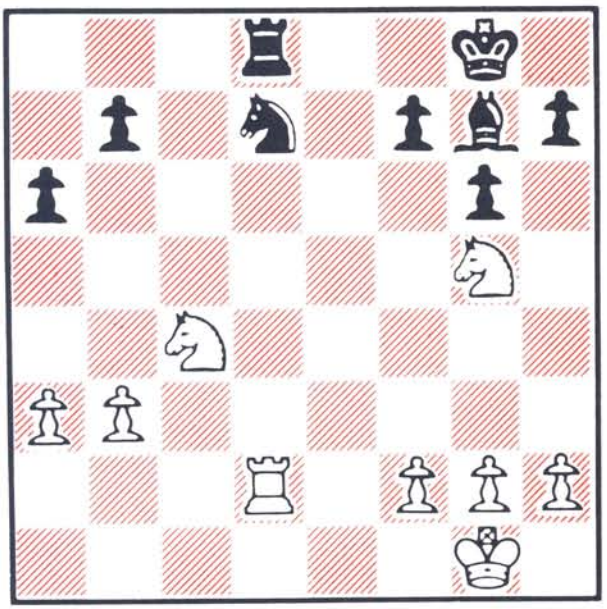


Figure 4. White to move

An additional item of interest concerns the program's desire to play RXQ, winning the queen (Figure 5). Assume that the program plays RXQ while performing a one ply search. The quiescence search tries R-K8 and finds that white is checkmated. The program then tries NXRP, PXN, RXQ. This position is reached in the quiescence search at a point where checks are not considered (considering all checks could result in an infinite loop). This is because there are stringent controls on what is included in the quiescence search in order to conserve time. The program happily evaluates this as good for white. After playing this, and receiving black's response, the program is surprised to find that once more it cannot play RXQ.

In computer chess tournaments, there have been many amusing examples of this problem. Suppose that black is faced with the loss of a trapped rook. There have been actual games played where black would force white to capture a series of pawns and pieces merely to delay (eliminate) the loss of the rook.

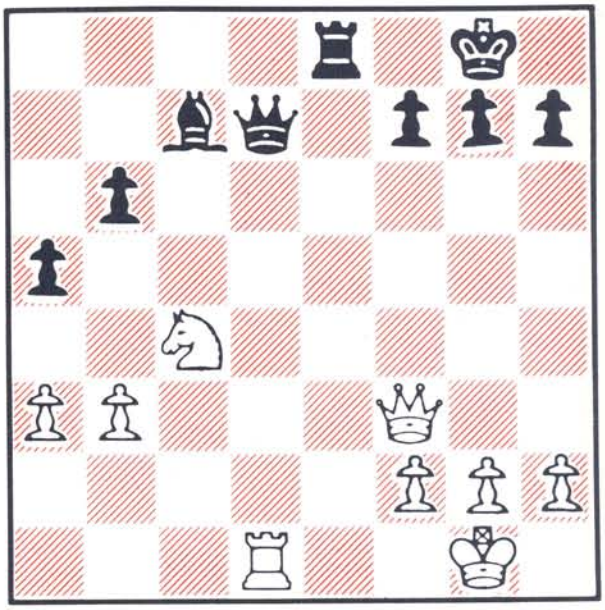


Figure 5. White to move

The quiescence search in CRAY BLITZ is far better than many other chess programs, but is still subject to an occasional attack of horizon effect. The solution seems to be in spending more time in the quiescence search to more accurately assess the tactical activity present. It turns out that this is far easier said than done, but progress is continually being made in this weakest area of computer chess.

A human understands that delaying the attacks does not eliminate them, but the computer, by forcing the attacks out of the search, thinks that it has totally avoided the problem.

POSITIONAL EVALUATION

It is always interesting to listen to chess masters comment on computer chess programs. The common assessment is that they are tactically brilliant but positionally weak. However, positional chess is really just long-range tactical chess. If the program can win material, the tree search is responsible for finding out how; or, if the program is threatened with losing material, the tree search is responsible for finding a defense. Positional chess is simply determining what to do when there is nothing to do.

In chess literature, there are a number of basic rules that beginners are cautioned to learn and remember, such as "control and occupy the center of the board", "occupy open files with rooks", and "keep the king safely tucked away behind pawns." In order to play reasonable chess, then, a computer must understand and follow these rules.

The dominating term in the evaluation function is material. That is, the program will almost always prefer capturing a tangible piece to win material rather than winning some positional

advantage while giving up a material advantage. As can be seen, the positional judgement of the program is only used when material balance cannot be altered.

The positional evaluators are really quite simple, although they are relatively long. For example, the chess principle "a knight on the rim is dim" implies knights should avoid squares on the edge of the board. If you imagine the chess board as an 8 x 8 array, the necessary loops to check each square on the outer edge of the array are quite simple. Each square must be scanned looking for a +/- 2 (white/black knight). For each one found, add a penalty for the correct side to the positional score (assuming + scores are good for white, a white knight on the edge might get -100 added to the positional score). If the tree search encounters this identical position, except that the knight has moved off of the edge, the positional score would be 100 points better, causing this position to be favored.

In chess literature, there are a number of basic rules that beginners are cautioned to learn and remember. In order to play reasonable chess, then, a computer must understand and follow these rules.

CRAY BLITZ has a very large number of these rules programmed into the evaluation function, and each is quite simple to understand. However, the whole collection is extremely comprehensive. The evaluation function is the principal reason that CRAY BLITZ plays chess at the level it does currently. Even International Grandmaster Edmar Mednis was extremely impressed with the program's positional play (he was also awed at the program's tactical play). □

—ABOUT the AUTHOR—

Robert Hyatt *is an Instructor and Chief of Systems at the University of Southern Mississippi in Hattiesburg. He received his B.S. in Computer Science from USM in 1970 and has remained there to teach and do research. Bob has been competing in computer chess tournaments with BLITZ since 1976. He has had CRAY-1 support from Cray Research since April of 1980. Recently, Bob completed work on a microprocessor-based electronic chess board which he uses in tournament play.*

In the next issue of CRAY CHANNELS, Bob describes a computer chess tournament and evaluates CRAY BLITZ.