THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Vol. II No. 3 $2.50 US

Heuristic Search in Hi-Q page 2

Build a High-Resolution S-100 Graphics Board
Part Two: Theory of Operation page 12

Multi-user:
Etherseries page 17

System Integration
Part Two: Disk Controllers and CP/M 2.2 System Generation page 21

New Products page 28

The Computerist's Calendar page 20

Editor's Page

The End of the "Programmer Prima Donna"

The computer software market is changing rapidly now that more computers are being used in offices and homes by non-computerists. Until recently most of the microcomputers were used by people interested in computers. These knowledgeable users were willing to make do with awkward programs or rewrite them, but the market is changing and has now reached a point where programs will have to be designed for the end user in order to be successful.

When microcomputers first became popular, only programmers knew enough about these marvelous new devices to foresee what they could do. These pioneering individuals wrote business programs, and we were amazed at the power available in word processors, data bases, and other useful programs. These early business programs were the primary reason for the rapid expansion of micro sales.

Unfortunately, the programmers (who were the only ones to understand the micro) did not understand the requirements of a business system. The result was that hundreds of programs were developed which were difficult to use, and which *almost* did the job. The programmers worked in isolation and produced elaborate programs which sold because they were so much better than working without a computer.

The programmers were "high priests" who decided what the customers should use, and the customers did not have enough experience with computers to intelligently evaluate the offerings. The programmers also lost sight of who their users were, and were not really interested in understanding their needs. The software industry talks about alpha and beta testing of programs, but this testing was (and perhaps still is) done by experienced computerists. The testing should really be done by the lowest level of people expected to use the program. If you are selling a program which will be purchased and used by other programmers it should be tested by programmers. If the program will be used by people on the street, it should be tested by people on the street. It took a long time for other industries to realize that they had to identify their market, really visualize the individuals expected to lay their cash on the counter and put the product to use, and then spend enough time in the users' environment to understand what they did in a typical work day. If you want to test a program for real estate offices, you should load the program and a computer into your car, and drive around until you find an office whose personnel represent the least knowledgeable segment of your market. Offer them whatever cash it takes to have them try your program using the computer you supply, give them the documentation, and then sit in the corner and watch what happens. If they have to ask you a question, or you have to point out something they are doing wrong, the experiment has failed and you had better go back and make revisions. You may also find that while the program is easy to understand and put to use, it just may not perform the required functions.

The problem of non-performing software became very evident to us while we were searching for a data base for use on our new CP/M machine. We had been using *The General Manager* by Sierra On-Line on our Apple® , and were quite satisfied with it, but we are changing to a S-100 system and need a simple data base. So far the offerings we have seen for CP/M are expensive, hard to use, and are incapable of handling our mail list needs. We also reviewed some mail list programs, but they failed to handle our needs for 3rd

continued on page 25

Editor/Publisher................... Art Carlson
Art Director................... Joan Thompson
Technical Editor................... Lance Rose
Technical Editor..................... Phil Wells
Production Assistant........... Judie Overbeek
Contributing Editor............. Ernie Brooner

The Computer Journal® is published 12 times a year. Annual subscription is $24 in the U.S., $30 in Canada, and $39 in other countries.

Entire contents copyright © 1984 by The Computer Journal.

Postmaster: Send address changes to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquires to: The Computer Journal, P.O. Box 1697, Kalispell, MT 59903-1697.

HEURISTIC SEARCH IN HI-Q

by Henry W. Davis
Computer Science Department
Wright State University, Dayton, Ohio

Computer games challenge humans to be smart. Have you ever wanted to reverse the role? This article is about how to write a program which makes the computer appear as smart as a human, or smarter! It also demonstrates a technique called "heuristic search," an important component in many artificial intelligence systems.

Our medium is "hi-Q," a popular puzzle played in a certain chain of restaurants by customers who are waiting for their food. When I first encountered hi-Q, I was so intrigued that I decided to share it with my students in an artificial intelligence course at Wright State University. The course includes heuristic search algorithms. On two occasions, I asked students to try these algorithms on hi-Q. They used a wide variety of languages and computers, from BASIC, FORTRAN, and PASCAL on home computers to SIMSCRIPT on a CYBER. They obtained a number of fascinating results showing that heuristic programs can do quite well at hi-Q, considerably better than most humans.

In this article, I will describe the basic algorithm used along with specifics of the hi-Q environment. Then I would like to share with you some aspects of a particularly nice program written by Ed Dudzinski, a former masters degree student in computer science at Wright State. Ed currently works on software optimization for array processors at the Wright-Patterson Air Force Base in Dayton, Ohio. His program, written in FORTRAN, is interesting because it performs well while demanding little memory—less than 40,000 bytes in a home computing environment. Nevertheless it has analyzed game situations involving as many as 190,000 different board configurations.

Our results are only a beginning. It is still not clear what the best computer hi-Q strategy is, even for the simplest hi-Q version described below.

The Rules of Hi-Q

A common version of hi-Q has 21 holes drilled into a piece of wood. Figure 1 shows the pattern: there is assumed to be one hole in the center of each of the numbered squares. The puzzle begins with pegs in 20 of the holes. The goal is to "jump and remove" 19 of these pegs, ending with only one peg on the board. The rules are as follows:

1) A peg can be moved only by jumping and thereby removing a single adjacent peg.

2) A peg can jump either horizontally (eg., from hole 5 to hole 7), vertically (eg., hole 17 to 7), or diagonally (eg., hole 7 to hole 15).

3) The jump can be made only if the destination is empty (eg., hole 7 in a 5-to-7 jump) and the jumped hole (hole 6) is full; the jumped peg is removed.

Figure 2 shows the first three moves of a typical game. This game was initialized by arbitrarily choosing hole 6 to be empty. The game is easily played and studied by simply drawing a big version of Figure 1 (without the numbers) and using pennies instead of pegs. In fact, for this reason it is sometimes called the "20-penny puzzle."

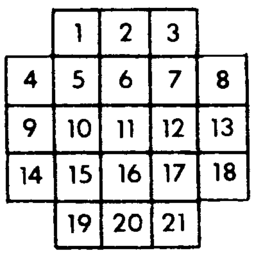


Figure 1: Hi-Q is played on a board with positions patterned as shown above. In the center of each numbered square is a hole for a single peg. Initially 20 pegs are placed randomly into the holes. The pegs may jump one another horizontally, vertically or diagonally as long as they land in a vacant location. Jumped pegs are removed from the board and the goal is to remove all but one peg.

The version just described will stump most people for quite a while. Try it! For the very ambitious there are harder versions coming up.

The Basic Algorithm

The basic hi-Q search algorithm is a variation of Nils Nilsson's "ordered search algorithm" which may be found in Chapter 3 of his 1971 book **Problem solving Methods in Artificial Intelligence.** This is also the "graphsearch procedure in Chapter 2 of his 1981 book **Principles of Artificial Intelligence.** It is shown in Figure 4 and explained below.

The many possible configurations of the hi-Q board are called *states*. The states are connected by arrows, or directed arcs, which represent legal moves transforming one state into another. The result is a directed graph, called the *state space*. Figure 3 shows a very small portion of the state space for the hi-Q puzzle whose initial state is Figure 2a.

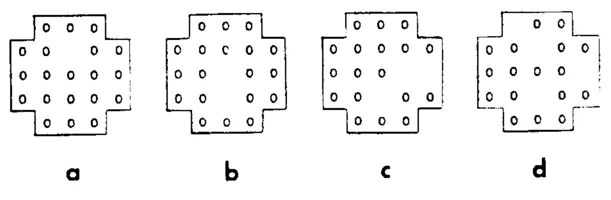


Figure 2: A typical initial state for hi-Q is shown in Figure 2a. Any square could be left empty, but in this case it's square number 6 (using the numbering scheme of Figure 1). Figures 2b, 2c, and 2d show the results of typical first, second and third moves. For example, in the third move, the peg at 1 jumps the peg at 6 and lands in location 12.

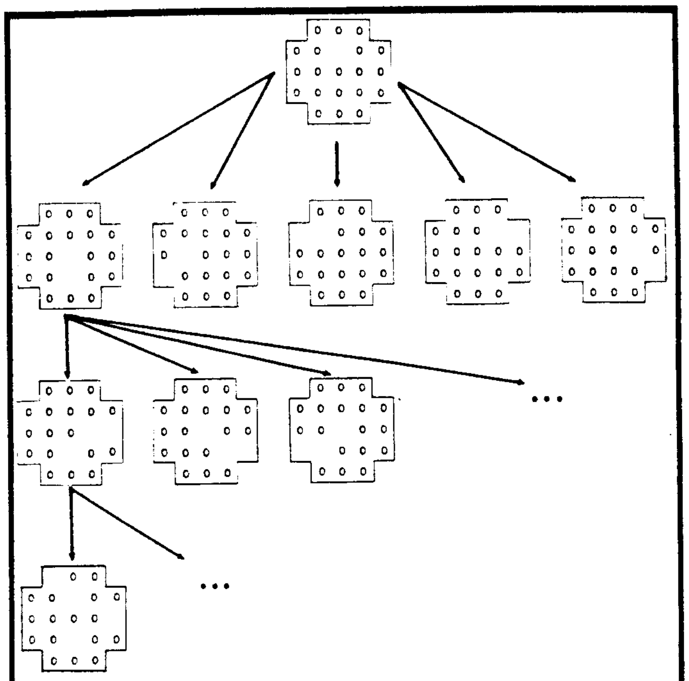


Figure 3: A very small portion of the "state space" of a hi-Q puzzle is shown. The different board configurations are called **states** or **nodes**. Arrows indicate legal moves from one state to another. The whole construct of states and legal moves forms a directed graph, called the **state space**.

```
Compute f (START), store it with START, and insert START into Open
CURRENT←NULL; FOUND←'FALSE'
DO UNTIL OPEN is empty or FOUND = 'TRUE'
    CURRENT← node on OPEN with least f value
            (resolve ties lexically)
    Insert CURRENT into CLOSED and delete it from OPEN
    Generate all successors to CURRENT
    IF some successor is a goal THEN
        Report solution (using search tree)
        FOUND←'TRUE'
      ELSE
        For each successor M of CURRENT DO
          If M is not on OPEN or CLOSED THEN
            Compute f(M) and store it with M
            Direct a "parent pointer" from M to CURRENT
                (for search tree)
            Insert M into Open

IF FOUND = 'FALSE' THEN report failure.
```

Figure 4: The basic hi-Q search algorithm starts by putting the initial puzzle configuration on the OPEN list. In the main iteration a node is removed from OPEN and its successors are generated by applying all legal moves. If a goal (only one peg left) is not found, then those successors not previously seen are added to OPEN. If a goal is found, then search tree pointers enable the program to report the route it discovered.

The algorithm "knows" the legal moves and takes as input an initial state. It performs all legal moves on the initial state, generating new states. The legal moves are then performed on some or all of these new states obtaining still more states, etc. Intuitively, this process has the program "wandering around the state space" looking for a goal state—much like a rat in a maze. Obviously if this is to work, the program needs some bookkeeping devices to keep track of where it has been, how it got there, and where it might still go. It also needs a decision mechanism to determine where to go next. The key ingredients for doing this are the *open list*, the *closed list*, the *search tree*, and the *evaluation function*, each described below.

1) OPEN is a list of states, or nodes, in the state space which the program knows about but to which the legal moves have not yet been applied. Initially the beginning puzzle configuration is placed on OPEN. In a typical cycle of the search procedure a node is removed from OPEN and all legal moves are applied to it obtaining a set of *successor nodes*. One says that the node removed from OPEN was *expanded* and that the successor nodes were *generated* from it.

2) CLOSED is a list of nodes which have been removed from OPEN and expanded. Why bother to remember a node we have already expanded and hence seem to be through with? The reason is so that whenever we generate a new node we can tell whether or not it has been previously generated. Namely, we compare the new node with the entries on OPEN and CLOSED. If we find a copy of it there, then we will behave differently than we would had the node not been previously discovered.

3) When a new node is generated, the program will place in its record of the node a pointer to the node's parent. The resulting structure of nodes and pointers forms a tree, called the search tree. The initial problem state is the tree's root. The sole purpose of the search tree is to enable the program to find its way back to the start once it has found the goal. By following the parent pointers back to the root, a listing of the legal moves from start to goal is obtained.

4) We come now to the crucial question: In what pattern shall the program "move through the state space?" This is equivalent to asking "in what order shall it expand the nodes which are on OPEN?" This decision is made by an evaluation function, f. When f is evaluated on a puzzle state it returns a real number: the smaller that number is, the more likely it is felt that the given state is close to a goal state. The search program expands that node on OPEN whose f value is lowest. Most of the apparent "smartness" of our program is due to f. Without a good evaluation function, all is lost.

Let us examine the algorithm of Figure 4 more closely. In the outside iteration, the program removes from OPEN the "most promising node" (lowest f value), puts this node on CLOSED and then expands it, obtaining all legal successors. If there is no solution, then those nodes not seen before are placed on OPEN, and another iteration is made. CURRENT is a location which references the node now being expanded. If several nodes on OPEN are tied with the lowest f value, then the "lexically" lowest is chosen. This simply means that the program views the board description as a string of

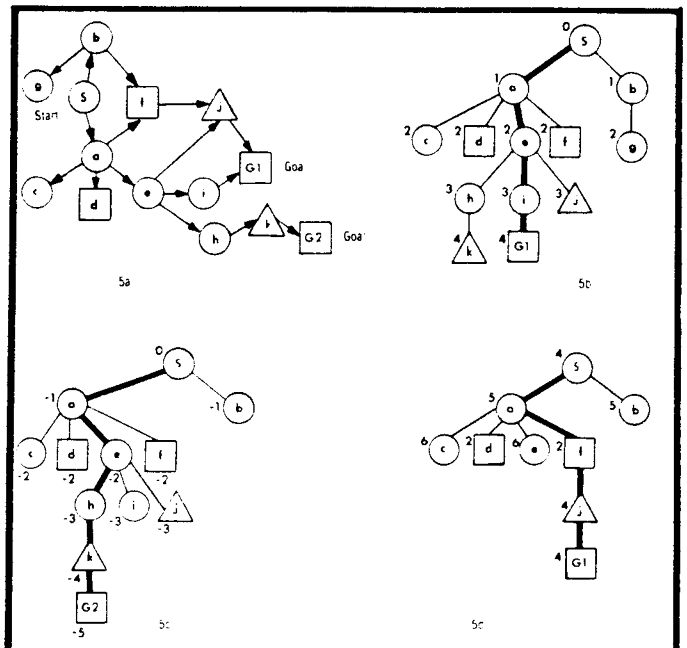


Figure 5: The state space for a fictitious puzzle is shown in Figure 5a. Figures 5b, 5c, and 5d show the search tree and solution (darkened path) found by three evaluation functions. In 5b, f rewards being close to START, yielding a breadth first search. In 5c, f rewards being far away from START, yielding a depth first search. In 5d, the f of 5b has added to it a heuristic component which punishes nodes for being circular, less for being triangular and not at all for being square. The numbers beside each node are the f values (see article for computation details). It is assumed that ties are broken lexically (eg., in figure 5b node a is expanded before node b).

characters and chooses the smallest string relative to normal sorting of character strings. Alternatively, such ties could be broken "arbitrarily." But it turns out that the success of many evaluation functions on a particular initial puzzle varies tremendously (sometimes by a factor of 1000) with how such ties are broken. Rather than leaving the tie-breaking up to an accident of coding, we choose to make it an explicit part of the algorithm. For one thing, our results are more easily duplicated by others. We return to this problem later because it raises the question of how we can properly judge the effectiveness of an evaluation function.

Evaluation Functions: The Main Source of Intelligence

It is important to understand how different evaluation functions can alter the search pattern. Figure 5 illustrates this. In Figure 5a the state space for a fictitious puzzle is shown. Some of the states are shaped as circles and others as triangles or squares. Each state is given a name (eg., S,a,b, or G2) and there are two goal states. For each node N in Figure 5a let g(N) be the least distance from S to N that the program has seen at a given instance, where we assign to each directed arc a distance of 1. One common practice is to set f(N) = g(N). Figure 5b shows the search tree that results when the algorithm of Figure 4 is run on this example. The darkened path shows the particular solution which is discovered. The numbers beside each node give its f value. As the algorithm iterates, the search tree is built up level-by-level, left-to-right. A completed level in the search tree corresponds to having investigated all nodes within a fixed distance of START in the state space. Due to this systematic expansion from START in all directions, the algorithm is called "breadth first." It is an example of a "blind search" because no heuristics are used.

Another type of blind search is called "depth first" because it always favors expanding those nodes which are furthest away from START. To achieve it in the puzzle of Figure 5a, set f(START) = 0; whenever a node Q is generated from a node P, we set f(Q) = f(P) - 1. The resulting search tree and solution obtained is shown in Figure 5c. A deeper and more costly goal is found than in the breadth first case; however, fewer nodes are generated. For hi-Q it is very important to go deep quickly. More about this shortly.

Now suppose we would like to have a more-or-less breadth first search tempered by some heuristic information. For example, suppose a hunch tells us that whenever the program generates a square node (eg., like d or f in Figure 5a) then it is probably close to a solution and it should keep moving in that direction. A triangular node (like j in Figure 5a) is similar but not as good. Then we might define the "heuristic function" h by:

h(N) = 0 if N is square
h(N) = 1 if N is triangular
h(N) = 4 if N is round

The heuristic function punishes nodes for being round or triangular, but the latter less. Define f = g + h. The effect of g is to "add breadth to the search." If the program gets carried away following square and triangular nodes deeply into the state space, then sooner or later g will grow and force it back to shallower nodes which it ignored earlier. In Figure 5d this f is applied to the puzzle of Figure 5a. It happens to work very well: the least cost goal is found while fewer nodes are generated or expanded than with the other evaluation functions. Setting f = h and dropping the g out works just as well in this case. In general, however, conventional wisdom suggests leaving the g component in for the reasons mentioned above: h might sometimes lead one astray and g helps cover for that event. (It's easy to alter Figure 5a so that this happens.)

The Hi-Q Landscape

Since the state space for hi-Q is finite, even a blind search, like breadth first, will eventually find a goal. The solution it reports will require only 19 moves because all solutions require exactly 19 moves. On what basis, then, can we say that one computer search heuristic is better than another? There are several criteria which students in my classes have used:

a) The number of nodes generated: An algorithm which consistently generated fewer nodes then others would seem to be working less hard.

b) The number of nodes expanded: When a node is expanded the algorithm is saying, "I think the goal is in this direction." If only 19 nodes are expanded, then the program went directly to the solution. I've never seen a human do that. An algorithm which consistantly scores in the low twenties in this area would have to be considered good.

c) CPU time: A low value is good while a high value offsets good performance in the first two areas.

d) Consistency: A good algorithm performs well for all initial states.

As we noted earlier, the performance of an evaluation function varies tremendously on a given initial puzzle when the f ties on OPEN are broken differently. A related phenomenen is that the same algorithm will often get wildly different answers when working on "symmetric duplicates." One configuration is a symmetric duplicate (SD) of another if it may be mapped into the other by a combination of 90° rotations and reflections on the diagonals. The initial configurations of Figure 6 are all SDs. In row two of Figure 3, states three and four are SDs. My students and I were initially surprised to see the same algorithm generate 100 nodes to solve one puzzle and then 100,000 nodes to solve one of its SDs. This will happen even if ties are broken lexically. The reason is very simple: most people code their programs so that symmetric holes are assigned different numbers and these numbers affect the order in which successors are generated. Thus the successors of two SDs will usually be put on OPEN in a different order. The effect is that ties are broken differently when the program works on two unequal SDs. Lexical breaking of ties does not help because the fact that one state is lexically smaller than another is no guarantee that an SD of the one state will be lexically smaller than an SD of the other.

There are several possible approaches to this problem. One is to collect statistics in categories (a), (b), (c), and (d), above, for all 21 initial puzzle configurations. This method was used to evaluate the Dudzinski program described below. A very elegant approach was taken by Joel W. Arnold: the order in which successors to a node are generated depends only on the class of SDs to which the node belongs and not on which particular SD it is. SD nodes generate SD successors in a corresponding order. His program behaves identically with respect to (a), (b), and (c) when input states are SDs. (Arnold's SDs are relative to rotation, but not to reflection.)

It is common to include a breadth component g (discussed above) in the evaluation function of puzzles. My students have universally found that in hi-Q this is harmful. On the contrary, a depth component is required. Without this, the search tree becomes too wide and programs run out of memory, or time. The evaluation functions we use have the form $f = d + h$, where h is the heuristic component and d forces depth, much the same way that h forces breadth. In hi-Q a natural form for d is $d(N) =$ number of pegs left in the configuration N.

Heuristics and Results

Five search techniques are compared in Ed Dudzinski's program. All use evaluation functions of the form $f = d + h$,

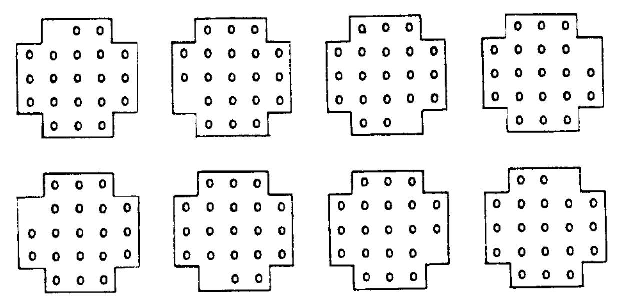


Figure 6: The eight initial configurations shown are "symmetric duplicates" of each other. The puzzles on each row may be obtained by successive rotations of 90 degrees. The puzzles in row 2 may be obtained from the ones above them via a diagonal reflection. Puzzles with more than one peg missing can also be SD's. Treating SD's "equally" requires careful coding.

where d is the depth component discussed above (number of pegs on board) and h is a heuristic function. All searches are depth first. This is achieved by requiring that each h be less than one in absolute value and never change sign. Thus the dominant rule in choosing nodes from OPEN is "take the deepest;" only if there are depth ties is h relevant. Further ties are resolved lexically. The five heuristic functions compared are as follows:

1) Blind depth search; h is zero. This gives a basis for comparison of the other heuristic functions.

2) Avoid filled corners; h is (number of filled corners)/10. The corner holes are numbered 1, 4, 14, 19, 21, 18, 8, and 3 in Figure 1. This heuristic punishes states with filled corners. The 1/10 factor keeps h less than unity assuring a depth first search.

3) Favor a filled center; h is −(number of filled holes)/10. Nodes are rewarded to the extent that their nine centermost holes are filled.

4) Favor filled pivotal spots; h is −(number of filled pivotal spots)/5. In Figure 1 locations 6, 10, 16, and 12 are called pivotal because more jumps can be made from and over them than any other hole.

5) Favor a high degree of freedom; h is −(number of possible next moves)/30. This heuristic says to move in such a way as to keep the number of options at a maximum. The maximum number of successors a node can have is about 18.

Ed ran the program on all 21 initial configurations using each of the five evaluation functions. Table 1 gives a summary of his results. The five algorithms are compared with respect to average number of nodes generated, average number of nodes expanded, and average CPU time required. To measure the consistency, or stability, of the algorithms, the standard deviation is also given. Table 2 gives the actual results for each algorithm and each possible start state. Ed evaluates the situation as follows:

The exhaustive depth-first search algorithm with lexical discrimination gives surprisingly good results, but this seems somewhat accidental. As algorithm 2 shows, and as other runs not included here have borne out, there are dead end branches that can easily trap an uninformed depth-first search into generating many thousands of nodes.

Results for algorithm 2 (avoid filled corners) are deceptive in the average. For all but three cases, this approach gives very good results with a small investment in CPU time. Unfortunately, one run caused the expansion of almost 190,000 nodes, greatly swelling the mean figures. Hence, stability was poor, to say the least.

The last three algorithms show more stability. Approach 3, favoring pegs in center holes, significantly improves on exhaustive depth-first search in all categories. Algorithm 4 (pivotal holes) improves on algorithm 3.

By far the most stable was the 5th algorithm (degree of freedom). Obviously, this approach will result in the generation of a lot of nodes, since it chooses the path of most successors: the average number of nodes generated by this approach exceeds the values for algorithms 3 and 4. However, the performance of algorithm 5 in terms of nodes expanded, or actual moves, is remarkable. Six times, out of the 21 starting configurations, the solution is found in the minimum 19 moves! In fact, for only two of the starting states does the program exceed 22 moves. The price for this facility of decision is paid in CPU time. Calculations of the number of possible next moves is considerably more time-consuming than methods used by the other algorithms.

The ideal heuristic function, if it exists, will combine the predictive power of algorithm 5 with the speed and simplicity of algorithm 4.

The Dudzunski Program

Many people who write successful hi-Q search programs do so on home computers. The Dudzinski program was written in CDC extended FORTRAN and run on the CYBER 750. The CYBER is a big machine but the program requires very little memory. For example, it allows space for only three hundred nodes to exist at a single time. Because of judicious space management, this has proven adequate for solving puzzles which generate over 189,000 nodes. The program has 220 executable FORTRAN statements and uses 9000 storage locations for arrays. In the CYBER the total lead module was 15,249 60-bit words. Upon converting this to a home computing environment, I estimate 39,000 bytes as a generous upper bound on space needs. This will be considerably reduced if the hi-Q board is represented more efficiently.

The terminology of the program is in terms of pennies instead of pegs. Each node has six fields:

a) 5 x 5 matrix to hold the board configuration (1 = penny, 0 = open, 9 = corner).

b) Open pointer.

c) Parent pointer.

d) Closed pointer.

e) F value, where F is the evaluation function.

f) Number of pennies on the board.

"Pointer" means the node number of another node. Three hundred nodes are created at load time via six arrays corresponding to each of a,...f, above. The FORTRAN declaration creates BOARD (5,5,300), OPENL(300), PARENTL(300), CLOSEDL(300), F(300), and NR PENNY(300). Any particular node "straddles" these arrays. For example, node 23 has its F value stored in F(23), the number of pennies on its board in NR PENNY(23) and the actual board configuration stored in the locations associated with BOARD(, ,23). If node 18 generates node 23, then the value in PARENTL(23) is 18.

OPEN and CLOSED are maintained as linked lists; i.e., as nodes chained together via the pointer values in fields b and d, above. Removing a node from OPEN and putting it on CLOSED means simply changing a few values in fields b and d. The nodes in OPEN are kept sorted on F value, and lexically where F values are tied. Thus the main routine always chooses the first node on OPEN for expansion.

To save time, puzzles with more than seven pennies removed are not checked for duplicates on OPEN or CLOSED. This is the only deviation from the algorithm of Figure 4. The program checks for symmetric as well as perfect duplicates for the first three moves and perfect duplicates for the next four.

To conserve memory, the following management device is used. Suppose the previous node expanded has five pennies on the board and the current

	Number of nodes generated		Number of nodes expanded		CPU time (sec)	
	average	standard deviation	average	standard deviation	average	standard deviation
1. Blind	372.4	489.8	280.9	498.0	.240	.188
2. Avoid Corners	9506.2	40204.8	9384.7	40208.7	3.893	16.068
3. Favor Center Holes	205.4	170.5	95.6	176.7	.174	.073
4. Favor Pivotal Holes	160.4	168.3	93.5	164.4	.155	.061
5. Maximize Freedom	214.5	44.1	26.2	23.9	.310	.147

Table 1: Five search algorithms are compared on all 21 initial hi-Q states. The first is a blind depth first search to give some basis of comparison for the others. The others are informed depth first searches. The average performance with respect to nodes generated, nodes expanded, and CPU time is given. In each case the standard deviation is given to measure the consistency of the algorithm. Algorithm 5 performs best in terms of number of nodes expanded and is very consistent. Unfortunately its CPU time is high.

one has eight. Since the search is depth first, the only way this could happen is for the previous node to have been a dead-end, i.e., no successors. The program is effectively backing up to a node three levels higher. There are no nodes on OPEN with five, six, or seven pennies. Therefore we can "free" the previous node and three generations of its ancestors without danger of eliminating a node involved in an eventual solution. As long as such freedom candidates are below the first seven levels, they are removed from CLOSED and placed back on a list of available nodes. This is how Dudzinski's program is able to solve puzzles requiring the generation of 189,000 nodes while there is only space for three hundred at a time.

Figure 7 shows the subprogram structure. The main routine (listing 1) first initializes the board and various lists by calling INIT. INIT reads the input puzzle and puts the first node on OPEN (see listing 2). PENNY loops through the open list calling PRUNE if the current node has no fewer pennies than the previous one (indicating a dead-end). Nodes are removed from OPEN, then placed on CLOSED if they are no more than seven deep. MOVE is called to generate successor nodes, check for a solution and insert some or all of the successors onto OPEN.

GETNODE and FREE (listing 3) manage the list of available nodes. This list is chained together via the "open pointer field" mentioned earlier. GETNODE provides the caller with the node number of an available node and removes that node from the available list. FREE returns a node to the available list.

EVAL is passed a pointer to a newly created node and returns its F value. The five evaluation functions used are shown in listing 4. PRUNE (listing 5) replenishes the list of available nodes whenever a dead-end is encountered. This

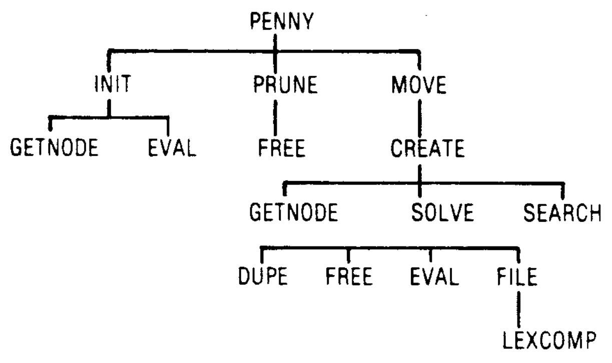


Figure 7: The calling structure of subroutines in the Dudzinski program is shown. PENNY initializes the board (via INIT) then removes nodes from OPEN, placing them on CLOSED as appropriate. MOVE is called periodically to generate successors, check for a solution and insert appropriate nodes onto OPEN. PRUNE frees nodes when dead-ends are encountered. EVAL evaluates f. The other module functions are described in the article.

	1. Blind search			**2. Avoid corners**			**3. Favor center**			**4. Favor pivots**			**5. Favor freedom**		
Initial Open Hole	nodes gen.	nodes exp.	CPU time	nodes gen.	nodes exp.	CPU time	nodes gen.	nodes exp.	CPU time	nodes gen.	nodes exp.	CPU time	nodes gen.	nodes exp.	CPU time
1	558	468	.321	147	28	.150	232	129	.192	109	27	.128	215	20	.279
2	130	42	.139	171	34	.160	154	52	.157	250	179	.172	213	22	.288
3	135	39	.145	143	22	.140	456	385	.282	110	33	.132	215	20	.287
4	125	21	.141	152	32	.161	155	19	.154	224	154	.185	215	20	.277
5	1182	1098	.547	189,257	189,153	75.718	107	22	.124	247	168	.199	201	34	.322
6	148	60	.154	877	780	.441	165	27	.160	86	20	.141	214	19	.279
7	125	21	.146	162	22	.163	108	22	.122	79	20	.135	185	22	.249
8	1182	1098	.542	173	33	.164	170	22	.166	80	19	.126	196	21	.263
9	143	60	.148	112	19	.137	107	19	.127	146	73	.136	214	19	.284
10	245	149	.196	119	23	.149	119	28	.137	84	20	.136	211	22	.323
11	154	63	.153	153	29	.139	184	30	.156	82	20	.122	199	19	.256
12	161	58	.157	3,407	3,277	1.496	157	19	.156	78	20	.128	174	19	.243
13	117	19	.139	136	22	.137	109	19	.129	184	104	.154	405	132	.961
14	151	63	.151	150	29	.150	181	30	.154	79	20	.132	215	20	.278
15	557	468	.310	217	91	.170	124	28	.139	114	52	.130	188	22	.260
16	161	58	.172	3,407	3,277	1.508	157	19	.161	77	20	.126	202	19	.275
17	118	19	.137	137	22	.133	110	19	.132	243	174	.179	207	20	.283
18	131	32	.149	184	31	.171	180	109	.167	80	19	.131	215	20	.271
19	132	62	.148	131	20	.150	147	19	.145	79	19	.128	196	21	.271
20	2042	1981	.884	134	19	.131	881	786	.459	860	782	.411	210	19	.272
21	124	19	.159	261	115	.189	311	205	.227	77	20	.124	215	20	.282
max	2042	1981	.884	189,257	189,153	75.718	881	786	.459	860	782	.411	405	132	.961
Min	117	19	.137	112	19	.131	107	19	.122	77	19	.122	174	19	.243
Average	372.4	280.9	.240	9,506.2	9,384.7	3.893	205.4	95.6	.174	160.4	93.5	.155	214.5	26.2	.310
st. dev.	489.8	498.0	.188	40,204.8	40,208.7	16.068	170.5	176.7	.073	168.3	164.4	.061	44.1	23.9	.147

Table 2: This shows the performance of all five algorithms on all 21 initial states with respect to number of nodes generated, nodes expanded and CPU time. Minimum, maximum, average and standard deviation figures are also given. The hole numbvrs in column 1 refer to Figure 1. We see, for example, that algorithm 5 usually scores in the low 20's for nodes expanded, going almost directly to the solution. Algorithm 2 expanded 189,153 nodes for initial state 5 but only 22 for its symmetric duplicate, state 17.

was described earlier.

MOVE (listing 6) makes all possible next moves and for each one calls CREATE (listing 7) to generate the successor, check it for goal status and put it on OPEN, if appropriate. When a goal is found, SOLVED (listing 8) is called to print out the solution along with the number of nodes generated and expanded to reach it. The solution is a display of twenty successive board configurations from start to goal.

Newly created non-goal nodes are passed by CREATE to SEARCH (listing 9). If a node is no more than seven deep, SEARCH calls DUPE to compare it with nodes already on OPEN and CLOSED. DUPE checks for symetric duplication on the first three levels and perfect duplication thereafter (see listing 10). If a duplication is found, then SEARCH frees the node. Otherwise it is inserted into OPEN by FILE (listing 11), lowest F values first and ties broken lexically. LEXCOMP (listing 12) compares game states and tells which is lexically smallest.

The names and uses of all global variables are tabulated in listing 1.

Conclusions

Ed's results suggest how humans can play a better hi-Q game. I can't do the computations for algorithm 5 in my head very fast, but I do find that algorithm 4 improves my game. Ed's results also indicate that the best computer search strategy for hi-Q has yet to be found. Among the ideas that have not been thoroughly explored are these: a) weight the board squares in a graduated fashion that is more subtle than any of algorithms 2, 3, or 4; b) reward and punish certain clustering patterns like three colinear pegs or density around center of gravity, that are independent of particular board squares; c) change the strategy according to puzzle depth; d) combine strategies in a weighted fashion and then look for the best weights.

To what extent is it cost-effective to check for duplicates on OPEN and CLOSED? I don't know. Perhaps a good heuristic goes so directly to the solution that there is little need to worry about duplicates.

There is another unanswered question: What is the size of the hi-Q space?

There are several harder versions of the puzzle that people often prefer. Consider using pennies instead of pegs. Replace one of the pennies with a nickel and demand that the nickel be the last coin on the board. Harder still, demand that the nickel be left in the center. Another variation: use all pennies and demand that the last penny be in the center. Joel Arnold wrote a very nice search program to solve this problem. He worked backwards, depth first, from the goal and found the following heuristics effective: a) down to an intermediate depth, concentrate on filling the perimeter of the puzzle; b) in later stages give orthogonal moves precedence over diagonal moves; c) up to the last move, filling the goal spot is rewarded and emptying it is penalized. (The last move must empty it.) Another ruse: he sought any state symmetric to the desired goal. If the found state was not the goal, transformations were made on the sequence of moves, yielding a true solution. I'm not sure that this is "fair." It does improve average CPU time, and, unsurprisingly, the number of nodes generated.

The last point raises the question of what is a fair solution. In some sense we want generality because this is what a human who plays the puzzle uses. Any program which has 21 different procedures for each of the 21 start states would have to be rejected. There are at least two ways one could enforce generality and toughen the problem a bit. 1) Demand that the program perform well when given a partially solved puzzle; thus instead of 21 start states, there are thousands. 2) Don't tell the program until input time what the exact board shape is. It might be told upper bounds on size, but no more.

I hope some of you have as much fun with computer hi-Q as have my students and I.

Editor's Note: The following program listings in FORTRAN are slight modifications of those supplied by the author. A few changes were necessary in order to be compatible with plain vanilla FORTRAN. These changes involved substituting six-character variable names for some that were seven characters long, removing some extended assignment statements (e.g., A = B = C = 0), substituting single quotes for double quotes surrounding literal constants and removing message strings from STOP statements. Even so, the changes required were far fewer than converting one dialect of BASIC to another, which illustrates the advantage of programming in a reasonably standardized language.

We ran the programs listed here with Microsoft FORTRAN-80 under CP/M and obtained the same results listed by the author in Table (x), the only difference being that the CPU times were about two orders of magnitude greater. Lance Rose, Technical Editor

Listing 1

```
      PROGRAM PENNY
C  THE MAIN ROUTINE INITIALIZES VARIABLES AND LOOPS THROUGH
C  THE OPEN LIST, WHICH IS ORDERED BY F-VALUE.  SUBROUTINE
C  MOVE IS CALLED TO GENERATE SUCCESSORS TO THE CURRENT NODE.
C
C  GLOBAL VARIABLES:
C    COMMON NODE    -  ONE NODE OF THE GAME SEARCH TREE
C    BOARD          -  THE 5 X 5 PLAYING BOARD (CORNERS ARE UNUSED)
C    OPENL          -  LINKS NODES ON THE OPEN LIST, WHICH CONTAINS ALL
C                      UNCHECKED GAME STATES; THIS POINTER IS ALSO
C                      USED TO LINK THE LIST OF AVAILABLE NODES
C    PARNTL         -  LINKS A NODE TO ITS PARENT, WHICH IS A BOARD
C                      CONFIGURATION ONE MOVE PREVIOUS TO ITSELF
C    CLOSDL         -  LINKS NODES ON THE CLOSED LIST FOR DUPLICATE
C                      CHECKING; THE BOARD CONFIGURATIONS IN THE CLOSED
C                      LIST HAVE ALREADY BEEN EXPANDED
C    F              -  THIS IS THE VALUE OF THE EVALUATION FUNCTION WHICH
C                      ATTEMPTS TO PREDICT THE PROBABILITY OF A SOLUTION
C                      IN A NODE'S SUCCESSORS
C    NPENNY         -  THE NUMBER OF PENNIES REMAINING ON THE BOARD
C    COMMON LISTS   -  HEADS OF THE THREE LISTS OF NODES
C    HEAD           -  HEAD OF THE LIST OF AVAILABLE NODES
C    OPHEAD         -  HEAD OF THE LIST OF OPEN NODES
C    CLHEAD         -  HEAD OF THE LIST OF CLOSED NODES
C    COMMON CTRS    -  COUNTERS USED TO TRACK THE EFFECTIVENESS OF THE
C                      PROGRAM HEURISTICS AD MECHANICS
C    NRGEN          -  THE NUMBER OF NODES GENERATED
C    NREXP          -  THE NUMBER OF NODES EXPANDED
C    MAXNOD         -  THE MAXIMUM NUMBER OF NODE DATA AREAS IN USE AT
C                      ANY TIME
C  LOCAL VARIABLES:
C    CURRNT         -  POINTER TO THE NODE BEING EXAMINED FOR POSSIBLE
C                      EXPANSION
C    HLDPTR         -  POINTER OT THE NODE EXPANDED PREVIOUS TO THE
C                      CURRENT NODE
C    PASS1          -  LOGICAL SWITCH TO SKIP PRUNING TEST ON FIRST PASS
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
     & ,OPHEAD,CLHEAD,NRGEN,NREXP,MAXNOD
      REAL F
      INTEGER CURRNT,HLDPTR
      LOGICAL PASS1
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/LISTS/HEAD,OPHEAD,CLHEAD
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      DATA PASS1/.TRUE./
      CALL INIT
      HLDPTR=OPHEAD
 100  IF(OPHEAD.EQ.0) GO TO 900
      CURRNT=OPHEAD
      OPHEAD=OPENL(CURRNT)
      IF(PASS1) GO TO 200
C  IF THE SEARCH IS NO LONGER GOING DOWN INTO THE STATE SPACE,
C  THEN WE HAVE REACHED A DEADEND (ALL ALGORITHMS ARE VARIATIONS
C  OF DEPTH-FIRST) AND NODES WITH FEWER THAN 13 PENNIES ARE
C  PRUNED FROM THE SEARCH TREE TO FREE STORAGE.
      IF(NPENNY(CURRNT).GE.NPENNY(HLDPTR))
     &  CALL PRUNE(HLDPTR,NPENNY(CURRNT))
C  SINCE ONLY BOARDS GENERATED BY THE FIRST 7 MOVES ARE CHECKED
C  FOR DUPLICATES, NODES LOWER IN THE STATE SPACE ARE NOT SAVED
C  ON THE CLOSED LIST.
 200  IF(NPENNY(CURRNT).LT.13) GO TO 500
      CLOSDL(CURRNT)=CLHEAD
      CLHEAD=CURRNT
 500  CALL MOVE(CURRNT)
      HLDPTR=CURRNT
      PASS1=.FALSE.
      GO TO 100
 900  STOP 1
      END
```

Listing 2

```
      SUBROUTINE INIT
C  THE INITIAL BOARD CONFIGURATION IS GENERATED AND FILED ON OPEN.
C  "EVAL" IS CALLED TO COMPUTE ITS F VALUE. POINTERS ARE INITIALIZED
C
C  LOCAL VARIABLES:
C    PTR             -  POINTER TO NEWLY-CREATED INITIAL NODE
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
     & ,OPHEAD,CLHEAD,NRGEN,NREXP,MAXNOD
      REAL F
      INTEGER PTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/LISTS/HEAD,OPHEAD,CLHEAD
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      CLHEAD=0
      OPHEAD=0
      HEAD=1
      NREXP=0
      NRGEN=1
      DO 100 I = 1,299
 100  OPENL(I)=I+1
      OPENL(300)=0
      CALL GETNOD(PTR)
      DO 200 I = 1,5
      DO 200 J = 1,5
 200  BOARD(I,J,PTR)=1
      BOARD(1,1,PTR)=9
      BOARD(1,5,PTR)=9
      BOARD(5,1,PTR)=9
      BOARD(5,5,PTR)=9
C     READS THE INITIAL OPEN SQUARE FOR THIS GAME
      READ(1,300) I,J
 300  FORMAT(2I3)
      BOARD(I,J,PTR)=0
      NPENNY(PTR)=20
      OPENL(PTR)=0
      CLOSDL(PTR)=0
      PARNTL(PTR)=0
      OPHEAD=PTR
      CALL EVAL(PTR)
      RETURN
      END
```

Listing 3

```
      SUBROUTINE FREE(PTR)
C  DATA AREAS NO LONGER NEEDED ARE RETURNED TO THE LIST OF AVAILABLE
C  NODES.
C
C  PARAMETERS:
C    PTR             -  I/O:POINTER TO USED NODE BEING RETURNED TO THE
C                       LIST OF AVAILABLE NODES
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
     & ,OPHEAD,CLHEAD
      REAL F
      INTEGER PTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/LISTS/HEAD,OPHEAD,CLHEAD
      PARNTL(PTR)=0
      CLOSDL(PTR)=0
      NPENNY(PTR)=0
      F(PTR)=0.
      OPENL(PTR)=HEAD
      HEAD=PTR
      PTR=0
      RETURN
      END


      SUBROUTINE GETNOD(PTR)
C  AVAILABLE DATA AREAS FOR NEW NODES ARE RETURNED
C  TO THE CALLING ROUTINES.
C
C  PARAMETERS:
C    PTR             -  O:POINTER TO AVAILABLE NODE DATA AREA
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
     & ,OPHEAD,CLHEAD,NRGEN,NREXP,MAXNOD
      REAL F
      INTEGER PTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/LISTS/HEAD,OPHEAD,CLHEAD
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      DATA MAXNOD/0/
      IF(HEAD.EQ.300) STOP 2
      PTR=HEAD
      IF(PTR.GT.MAXNOD) MAXNOD=PTR
      HEAD=OPENL(PTR)
      RETURN
      END
```

Listing 4: The five elvaluation functions sampled are shown.

Listing 4a

```
      SUBROUTINE EVAL(PTR)
C  THE F-VALUE IS CALCULATED FOR ORDERING NODES ON THE OPEN LIST.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NEWLY-CREATED NODE
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
C  THE EVALUATION FUNCTION SIMPLY EQUALS THE NUMBER OF MOVES
C  TO THE SOLUTION, RESULTING IN A SIMPLE DEPTH-FIRST SEARCH.
      INTEGER PTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      F(PTR)=FLOAT(NPENNY(PTR))
      RETURN
      END
```

Listing 4b

```
      SUBROUTINE EVAL(PTR)
C  THE F-VALUE IS CALCULATED FOR ORDERING NODES ON THE OPEN LIST.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    CORNER          -  THE NUMBER OF CORNER PENNIES
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
C  BOARD CONFIGURATIONS WITH FEWEST CORNER PENNIES ARE FAVORED.
      INTEGER PTR,CORNER
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      CORNER=0
      IF(BOARD(1,2,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(1,4,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(2,1,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(2,5,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(4,1,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(4,5,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(5,2,PTR).EQ.1) CORNER=CORNER+1
      IF(BOARD(5,4,PTR).EQ.1) CORNER=CORNER+1
      F(PTR)=FLOAT(NPENNY(PTR))+FLOAT(CORNER)/10.
      RETURN
      END
```

Listing 4c

```
      SUBROUTINE EVAL(PTR)
C  THE F-VALUE IS CALCULATED FOR ORDERING NODES ON THE OPEN LIST.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    COUNT           -  THE NUMBER OF PENNIES IN THE 9 CENTRAL LOCATIONS
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
C  BOARD CONFIGURATIONS WITH PENNIES IN 9 CENTRAL LOCATIONS
C  ARE FAVORED.
      INTEGER PTR,COUNT
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COUNT=0
      DO 100 I = 2,4
      DO 100 J = 2,4
 100  IF(BOARD(I,J,PTR).EQ.1) COUNT=COUNT+1
      F(PTR)=FLOAT(NPENNY(PTR))-FLOAT(COUNT)/10.0
      RETURN
      END
```

Listing 4d

```
      SUBROUTINE EVAL(PTR)
C  THE F-VALUE IS CALCULATED FOR ORDERING NODES ON THE OPEN LIST.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    COUNT           -  THE NUMBER OF PENNIES IN THE 4 PIVOTAL LOCATIONS
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
C  BOARD CONFIGURATIONS WITH PENNIES IN 4 PIVOTAL LOCATIONS
C  ARE FAVORED.
      INTEGER PTR,COUNT
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COUNT=0
      IF(BOARD(2,3,PTR).EQ.1) COUNT=COUNT+1
      IF(BOARD(3,2,PTR).EQ.1) COUNT=COUNT+1
      IF(BOARD(3,4,PTR).EQ.1) COUNT=COUNT+1
      IF(BOARD(4,3,PTR).EQ.1) COUNT=COUNT+1
      F(PTR)=FLOAT(NPENNY(PTR))-FLOAT(COUNT)/5.0
      RETURN
      END
```

Listing 4e

```
      SUBROUTINE EVAL(PTR)
C  THE F-VALUE IS CALCULATED FOR ORDERING NODES ON THE OPEN LIST.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    COUNT           -  THE NUMBER OF SUCCESSORS OF THE PASSED NODE
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
C  THIS EVALUATION FUNCTION REDUCES THE ACTUAL F VALUE (MOVES TO
C  SOLUTION) BY A FRACTION PROPORTIONAL TO THE NUMBER OF CHILDREN
C  OF THE NODE, THUS FAVORING THOSE BOARD CONFIGURATIONS WITH THE
C  GREATEST NUMBER OF SUCCESSOR NODES.  BY CONVERTING THE NUMBER
C  OF CHILDREN INTO A FRACTION LESS THAN ONE (THE MAXIMUM NUMBER OF
C  SUCCESSOR NODES IS < 30), WE RETAIN A DEPTH-FIRST SEARCH.
      INTEGER PTR,COUNT
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COUNT=0
      DO 100 I = 1,5
      DO 100 J = 1,5
        IF(BOARD(I,J,PTR).NE.1) GO TO 100
          IF(I.LT.3) GO TO 10
          IF(BOARD(I-1,J,PTR).EQ.1.AND.BOARD(I-2,J,PTR).EQ.0)
     &      COUNT=COUNT+1
 10       IF(I.LT.3.OR.J.GT.3) GO TO 20
          IF(BOARD(I-1,J+1,PTR).EQ.1.AND.BOARD(I-2,J+2,PTR).EQ.0)
     &      COUNT=COUNT+1
 20       IF(J.GT.3) GO TO 30
          IF(BOARD(I,J+1,PTR).EQ.1.AND.BOARD(I,J+2,PTR).EQ.0)
     &      COUNT=COUNT+1
 30       IF(I.GT.3.OR.J.GT.3) GO TO 40
          IF(BOARD(I+1,J+1,PTR).EQ.1.AND.BOARD(I+2,J+2,PTR).EQ.0)
     &      COUNT=COUNT+1
```

continued

Listing 4e, continued

```
40         IF(I.GT.3) GO TO 50
           IF(BOARD(I+1,J,PTR).EQ.1.AND.BOARD(I+2,J,PTR).EQ.0)
     &       COUNT=COUNT+1
50         IF(I.GT.3.OR.J.LT.3) GO TO 60
           IF(BOARD(I+1,J-1,PTR).EQ.1.AND.BOARD(I+2,J-2,PTR).EQ.0)
     &       COUNT=COUNT+1
60         IF(J.LT.3) GO TO 70
           IF(BOARD(I,J-1,PTR).EQ.1.AND.BOARD(I,J-2,PTR).EQ.0)
     &       COUNT=COUNT+1
70         IF(I.LT.3.OR.J.LT.3) GO TO 100
           IF(BOARD(I-1,J-1,PTR).EQ.1.AND.BOARD(I-2,J-2,PTR).EQ.0)
     &       COUNT=COUNT+1
100   CONTINUE
      F(PTR)=FLOAT(NPENNY(PTR))-FLOAT(COUNT)/30.0
      IF(COUNT.EQ.0) F(PTR)=10000.
      RETURN
      END
```

Listing 5

```
      SUBROUTINE PRUNE(PTR,HILEV)
C  NODES WITH FEWER THAN 13 PENNIES BEGINNING WITH THE PASSED NODE
C  AND BACK THROUGH ITS PARENT NODES TO THE PARENT WITH "HILEV"
C  NUMBER OF PENNIES ARE RETURNED TO THE LIST OF AVAILABLE NODES.
C  NODES WITH 13 OR MORE PENNIES REMAIN ON CLOSED FOR DUPLICATE COM-
C  PARISONS.  "HILEV" IS THE NUMBER OF PENNIES ON THE NODE CURRENTLY
C  BEING EXPANDED, WHICH IS WHERE OUR SEARCH HAS BACKED UP TO AFTER
C  REACHING A DEAD END.  BY PRUNING ONLY TO THE LEVEL OF THE CURRENT
C  NODE, WE DON'T CHANCE ELIMINATING PARENT NODES OF AN EVENTUAL
C  SOLUTION, BUT WE WILL ELIMINATE ALL DEADEND BRANCHES BELOW THE
C  13 PENNY LEVEL.
C
C  PARAMETERS:
C    PTR             -  I/O:POINTER TO NODE EXAMINED FOR POSSIBLE EXPAN-
C                       SION JUST BEFORE THE NODE BEING CURRENTLY EXAM-
C                       INED; THIS NODE COULD NOT BE EXPANDED; IF THIS
C                       NODE IS PRUNED, PTR IS SET TO ZERO
C    HILEV           -  I:THE NUMBER OF PENNIES ON THE GAME BOARD OF THE
C                       NODE CURRENTLY BEING EXAMINED FOR POSSIBLE
C                       EXPANSION
C  LOCAL VARIABLES:
C    PARENT          -  POINTER TO PARENT NODE
C    HLDPTR          -  POINTER USED TO TRACE FROM THE PASSED NODE BACK
C                       THROUGH ITS PARENTS
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
      REAL F
      INTEGER PTR,HILEV,PARENT,HLDPTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      IF(NPENNY(PTR).LT.HILEV) GO TO 100
      IF(NPENNY(PTR).GE.13) RETURN
      CALL FREE(PTR)
      RETURN
100   HLDPTR=PTR
      PTR=0
200   PARENT=PARNTL(HLDPTR)
      IF(NPENNY(HLDPTR).GE.13) RETURN
      CALL FREE(HLDPTR)
      IF(NPENNY(PARENT).GT.HILEV) RETURN
      HLDPTR=PARENT
      GO TO 200
      END
```

Listing 6

```
      SUBROUTINE MOVE(PTR)
C  ALL POSSIBLE NEXT MOVES ARE TAKEN, GENERATING EVERY SUCCESSOR
C  TO THE CURRENT NODE.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NODE BEING EXPANDED
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,NRGEN
     & ,NREXP,MAXNOD
      REAL F
      INTEGER PTR
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      NREXP=NREXP+1
      DO 100 I = 1,5
      DO 100 J = 1,5
        IF(BOARD(I,J,PTR).NE.1) GO TO 100
           IF(I.LT.3) GO TO 10
           IF(BOARD(I-1,J,PTR).EQ.1.AND.BOARD(I-2,J,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I-1,J,I-2,J)
10         IF(I.LT.3.OR.J.GT.3) GO TO 20
           IF(BOARD(I-1,J+1,PTR).EQ.1.AND.BOARD(I-2,J+2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I-1,J+1,I-2,J+2)
20         IF(J.GT.3) GO TO 30
           IF(BOARD(I,J+1,PTR).EQ.1.AND.BOARD(I,J+2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I,J+1,I,J+2)
30         IF(I.GT.3.OR.J.GT.3) GO TO 40
           IF(BOARD(I+1,J+1,PTR).EQ.1.AND.BOARD(I+2,J+2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I+1,J+1,I+2,J+2)
40         IF(I.GT.3) GO TO 50
           IF(BOARD(I+1,J,PTR).EQ.1.AND.BOARD(I+2,J,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I+1,J,I+2,J)
50         IF(I.GT.3.OR.J.LT.3) GO TO 60
           IF(BOARD(I+1,J-1,PTR).EQ.1.AND.BOARD(I+2,J-2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I+1,J-1,I+2,J-2)
60         IF(J.LT.3) GO TO 70
           IF(BOARD(I,J-1,PTR).EQ.1.AND.BOARD(I,J-2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I,J-1,I,J-2)
70         IF(I.LT.3.OR.J.LT.3) GO TO 100
           IF(BOARD(I-1,J-1,PTR).EQ.1.AND.BOARD(I-2,J-2,PTR).EQ.0)
     &       CALL CREATE(PTR,I,J,I-1,J-1,I-2,J-2)
100   CONTINUE
      RETURN
      END
```

Listing 7

```
      SUBROUTINE CREATE(PTR,L,M,N,O,P,Q)
C  THE PARENT NODE IS COPIED INTO A NEW DATA AREA, THE INDICATED
C  MOVE IS MADE, AND POINTERS ARE SET.  IF THE GENERATED BOARD
C  IS A SOLUTION, IT IS PRINTED OUT AND EXECUTION STOPS.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO NODE BEING EXPANDED
C    L,M             -  I:COORDINATES OF "JUMP FROM" POSITION
C    N,O             -  I:COORDINATES OF "JUMP-OVER" POSITION
C    P,Q             -  I:COORDINATES OF "JUMP TO" POSITION
C  LOCAL VARIABLES:
C    NEWPTR          -  POINTER TO NEWLY-CREATED SUCCESSOR NODE
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,NRGEN
     & ,NREXP,MAXNOD
      REAL F
      INTEGER PTR,NEWPTR,I,J,L,M,N,O,P,Q
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      CALL GETNOD(NEWPTR)
      NRGEN=NRGEN+1
      OPENL(NEWPTR)=0
      NPENNY(NEWPTR)=NPENNY(PTR)-1
      PARNTL(NEWPTR)=PTR
      DO 100 I = 1,5
      DO 100 J = 1,5
100   BOARD(I,J,NEWPTR)=BOARD(I,J,PTR)
      BOARD(L,M,NEWPTR)=0
      BOARD(N,O,NEWPTR)=0
      BOARD(P,Q,NEWPTR)=1
      IF(NPENNY(NEWPTR).NE.1) GO TO 150
      CALL SOLVED(NEWPTR)
      STOP
150   CONTINUE
      CALL SEARCH(NEWPTR)
      RETURN
      END
```

Listing 8

```
      SUBROUTINE SOLVED(PTR)
C  THE SOLUTION IS PRINTED OUT, ALONG WITH THE NUMBER OF NODES
C  GENERATED, THE NUMBER EXPANDED, AND THE MAXIMUM NUMBER OF NODE
C  DATA AREAS IN USE AT ONE TIME IN REACHING THIS SOLUTION.
C
C  PARAMETERS:
C    PTR             -  I:POINTER TO SOLUTION NODE
C  LOCAL VARIABLES:
C    PRTPTR          -  ARRAY OF POINTERS TRACING PATH FROM ROOT NODE
C                       TO SOLUTION
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,NRGEN
     & ,NREXP,MAXNOD
      REAL F
      INTEGER PTR,PRTPTR(20)
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/CTRS/NRGEN,NREXP,MAXNOD
      WRITE(1,1000) NRGEN,NREXP
      PRTPTR(1)=PTR
      DO 50 I = 2,20
      I1=PRTPTR(I-1)
50    PRTPTR(I)=PARNTL(I1)
      DO 200 K = 1,17,4
      I4=21-K
      I1=PRTPTR(I4)
      I4=20-K
      I2=PRTPTR(I4)
      I4=19-K
      I3=PRTPTR(I4)
      I4=18-K
      I4=PRTPTR(I4)
      WRITE(1,2000)
      DO 200 I = 1,5
200   WRITE(1,3000) (BOARD(I,J,I1),J=1,5),
     & (BOARD(I,J,I2),J=1,5),(BOARD(I,J,I3),
     & J=1,5),(BOARD(I,J,I4),J=1,5)
      WRITE(1,4000) MAXNOD
1000  FORMAT(1H1,I8,' NODES GENERATED',5X,I8,' NODES EXPANDED')
2000  FORMAT(1H )
3000  FORMAT(1H ,4(5I3,2X))
4000  FORMAT(1H0,'MAXIMUM NODES IN USE =',I6)
      RETURN
      END
```

Listing 9

```
      SUBROUTINE SEARCH(PTR)
C  BOARDS WITH 13 OR MORE PENNIES ARE CHECKED TO SEE IF DUPLICATES
C  EXIST ON THE OPEN OR CLOSED LIST AND FREED IF THAT IS THE CASE.
C  THOSE THAT ARE UNIQUE, AND ALL BOARDS WITH FEWER THAN 13 PENNIES
C  ARE FILED ON THE OPEN LIST IN INCREASING ORDER OF F VALUE.
C
C  PARAMETERS:
C    PTR             -  I/O:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    MOVPTR          -  POINTER USED TO TRAVERSE THE OPEN AND CLOSED LISTS
C
      INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
     & ,OPHEAD,CLHEAD
      REAL F
      INTEGER PTR,MOVPTR
      LOGICAL DUPE
      COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
     & CLOSDL(300),F(300),NPENNY(300)
      COMMON/LISTS/HEAD,OPHEAD,CLHEAD
      MOVPTR=OPHEAD
      IF(NPENNY(PTR).LT.13) GO TO 600
100   IF(MOVPTR.EQ.0) GO TO 300
      IF(NPENNY(PTR).NE.NPENNY(MOVPTR)) GO TO 200
      IF(.NOT.DUPE(PTR,MOVPTR)) GO TO 200
      CALL FREE(PTR)
      RETURN
200   MOVPTR=OPENL(MOVPTR)
```

continued

Listing 9, continued

```
      GO TO 100
300   MOVPTR=CLHEAD
400   IF(MOVPTR.EQ.0) GO TO 600
      IF(NPENNY(PTR).NE.NPENNY(MOVPTR)) GO TO 500
      IF(.NOT.DUPE(PTR,MOVPTR)) GO TO 500
      CALL FREE(PTR)
      RETURN
500   MOVPTR=CLOSDL(MOVPTR)
      GO TO 400
600   CALL EVAL(PTR)
      CALL FILE(PTR)
      RETURN
      END
```

Listing 10

```
        LOGICAL FUNCTION DUPE(PTR,TSTPTR)
C  A "TRUE" VALUE IS RETURNED IF THE PASSED NODES ARE DUPLICATE
C  CONFIGURATIONS.  BOARDS WITH 19,18, OR 17 PENNIES ARE CHECKED
C  FOR BOTH PERFECT AND SYMMETRIC DUPES (4 ROTATIONS THROUGH BOTH
C  MIRROR IMAGE REPRESENTATIONS).  BOARDS WITH LESS THAN 17
C  PENNIES ARE CHECKED ONLY FOR PERFECT DUPLICATES.
C
C  PARAMETERS:
C    PTR            -  I:POINTER TO NEWLY-CREATED NODE
C    TSTPTR         -  I:POINTER TO A NODE ON THE OPEN OR CLOSED LIST
C
        INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
        REAL F
        INTEGER PTR,TSTPTR,I,J
        COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
      & CLOSDL(300),F(300),NPENNY(300)
        DUPE=.TRUE.
        DO 100 I = 1,5
        DO 100 J = 1,5
  100   IF(BOARD(I,J,PTR).NE.BOARD(I,J,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        IF(NPENNY(PTR).LT.17) RETURN
        DUPE=.TRUE.
        DO 200 I = 1,5
        I1=6-I
        DO 200 J = 1,5
  200   IF(BOARD(I,J,PTR).NE.BOARD(J,I1,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 300 I = 1,5
        I1=6-I
        DO 300 J = 1,5
        J1=6-J
  300   IF(BOARD(I,J,PTR).NE.BOARD(I1,J1,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 400 I = 1,5
        DO 400 J = 1,5
        J1=6-J
  400   IF(BOARD(I,J,PTR).NE.BOARD(J1,I,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 500 I = 1,5
        DO 500 J = 1,5
        J1=6-J
  500   IF(BOARD(I,J1,PTR).NE.BOARD(I,J,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 600 I = 1,5
        I1=6-I
        DO 600 J = 1,5
        J1=6-J
  600   IF(BOARD(I,J1,PTR).NE.BOARD(J,I1,TSTPTR)) DUPE=.FALSE.
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 700 I = 1,5
        I1=6-I
        DO 700 J = 1,5
        J1=6-J
  700   IF(BOARD(I,J1,PTR).NE.BOARD(I1,J1,TSTPTR)) DUPE=.FALSE.
```

Listing 10, continued

```
        IF(DUPE) RETURN
        DUPE=.TRUE.
        DO 800 I = 1,5
        DO 800 J = 1,5
        J1=6-J
  800   IF(BOARD(I,J1,PTR).NE.BOARD(J1,I,TSTPTR)) DUPE=.FALSE.
        RETURN
        END
```

Listing 11

```
        SUBROUTINE FILE(PTR)
C  NODES ARE FILED ON THE OPEN LIST ACCORDING TO F-VALUE, LOWEST FIRST.
C  LEXICAL COMPARISONS ARE USED TO BREAK TIES BETWEEN F VALUES.
C
C  PARAMETERS:
C    PTR            -  I:POINTER TO NEWLY-CREATED NODE
C  LOCAL VARIABLES:
C    MOVPTR         -  POINTER USED TO TRAVERSE OPEN LIST, POINTS TO
C                      THE NODE CURRENTLY BEING COMPARED TO THE NEW NODE
C    LAGPTR         -  POINTER ALSO USED TO TRAVERSE OPEN LIST, POINTS
C                      TO THE NODE LAST COMPARED TO THE NEW NODE
C
        INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY,HEAD
      & ,OPHEAD,CLHEAD
        REAL F
        INTEGER PTR,MOVPTR,LAGPTR
        LOGICAL LEXCMP
        COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
      & CLOSDL(300),F(300),NPENNY(300)
        COMMON/LISTS/HEAD,OPHEAD,CLHEAD
        IF(OPHEAD.NE.0) GO TO 100
        OPHEAD=PTR
        RETURN
  100   IF(F(PTR).GT.F(OPHEAD)) GO TO 200
        IF(F(PTR).EQ.F(OPHEAD).AND.LEXCMP(OPHEAD,PTR))
      &  GO TO 200
        OPENL(PTR)=OPHEAD
        OPHEAD=PTR
        RETURN
  200   LAGPTR=OPHEAD
        MOVPTR=OPENL(OPHEAD)
  300   IF(MOVPTR.EQ.0) GO TO 400
        IF(F(PTR).LT.F(MOVPTR)) GO TO 400
        IF(F(PTR).EQ.F(MOVPTR).AND.LEXCMP(PTR,MOVPTR))
      &  GO TO 400
        LAGPTR=MOVPTR
        MOVPTR=OPENL(LAGPTR)
        GO TO 300
  400   OPENL(PTR)=MOVPTR
        OPENL(LAGPTR)=PTR
        RETURN
        END
```

Listing 12

```
        LOGICAL FUNCTION LEXCMP(PTR1,PTR2)
C  IF PTR1'S NODE HAS A BOARD CONFIGURATION THAT IS LEXICALLY LESS
C  THAN PTR2'S, "TRUE" IS RETURNED.  OTHERWISE, THE FUNCTION
C  RETURNS "FALSE".
C
C  PARAMETERS:
C    PTR1           -  I:POINTER TO NEWLY-CREATED NODE
C    PTR2           -  I:POINTER TO NODE ON OPEN LIST WITH AN F VALUE
C                      IDENTICAL TO THAT OF THE NEWLY-CREATED NODE
C
        INTEGER BOARD,OPENL,PARNTL,CLOSDL,NPENNY
        REAL F
        INTEGER PTR1,PTR2
        COMMON/NODE/BOARD(5,5,300),OPENL(300),PARNTL(300),
      & CLOSDL(300),F(300),NPENNY(300)
        DO 100 I = 1,5
        DO 100 J = 1,5
        IF(BOARD(I,J,PTR1).GT.BOARD(I,J,PTR2)) GO TO 150
  100   IF(BOARD(I,J,PTR1).LT.BOARD(I,J,PTR2)) GO TO 200
  150   LEXCMP=.FALSE.
        RETURN
  200   LEXCMP=.TRUE.
        RETURN
        END
```

■

Build a High-Resolution S-100 Graphics Board

Part Two: Theory of Operation

by Lance Rose, Technical Editor

In the first part of this series we saw how the video monitor uses its sweep circuits to create a raster scan with which text or graphics information can be displayed. In this installment I'll explain how the video board operates in order to send the desired information to the video display device.

Figure 1 contains a block diagram of the graphics circuit. Let's look at each block and its function in turn. The state generator contains the state information for each of the possible logic states that the board can have (more about "states" in a moment). It is essentially the brain of the board and all the other parts center around it. The scanning block continuously scans the video RAM and extracts information a byte at a time for display. The video output circuit combines the scanned information with timing information provided by the state generator and outputs a composite video waveform of the proper amplitude and impedance. The arbitration circuit controls when the video RAM may be accessed by the system for updating or reading information stored in it. And finally, the bus interface circuit contains the necessary buffers and control lines to interface the graphics board to the S-100 bus.

Before looking at the actual schematic, we need to understand a bit about what a "state" is. If this is redundant to you advanced builders, please bear with me for a moment.

Intuitively one might think that a state is a set of conditions for a circuit which has all signal levels and timings specified, and in fact that is pretty much it. What we are using here is called a "state machine" which is, in simple terms, just a ROM or set of ROMS where each memory address of the ROM(s) causes the pertinent information for that state to be output on the data lines. This pertinent information is:

(1) how long the current state lasts,

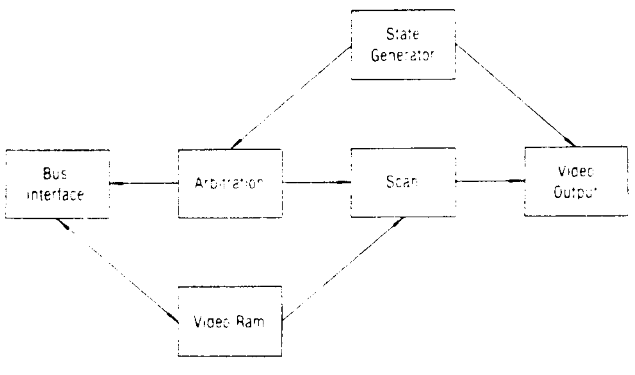


Figure 1: Block diagram of the graphics circuit.

(2) the necessary output signals for this state in order to control the rest of the circuit, and

(3) what the next state will be.

Complicated state machines can perform things like looping and branching to other states (microprocessors are examples of complicated state machines) but here the circuit is simplified by assuming the next state to always be one address higher in the ROM than the previous state. This does away with looping and branching (not needed here anyway) as well as additional ROM space to store the address of the next state. This method is analogous to the program counter in a microprocessor which simply fetches its next instruction from the next higher memory address unless a branch occurs in the program.

When the highest address of the ROMs is reached, the address counters simply "wrap around" to the first state again. Since we're dealing with a full interlaced video frame, this wraparound occurs every 1/30 of a second.

Let's go through the circuit in Figure 2 and look at how the various parts function.

Two of the gates in U1 are arranged in a conventional crystal oscillator with a third gate serving as a buffer. This oscillator is known as the "dot clock" since a new dot on the horizontal line is displayed with each clock cycle. The 16MHz value is chosen as a frequency which will be compatible with most video monitors, i.e. will not cause the display to go off the edge of the screen but will give a full screen display. If insufficient adjustment is available in the monitor to view the entire horizontal extent of the display, a higher frequency crystal can be used and the values in the state ROMs changed. Conversely, a slower clock will widen the horizontal display. Another choice is to program the ROMs to display less than 640 horizontal dots. One nice thing about a state machine is that it can be reprogrammed to tailor the board to different systems. It is possible to be compatible with some foreign TV standards by doing this. I will discuss this in more detail later on.

Since the timing is critical in the scanning portion of the circuit, gate delays have to be taken into account. This is done by providing a second dot clock signal which is nominally 180° out of phase with the first. This allows us to use either of two clock signals which differ by about 31nsec. This figure is just about right to compensate for the additional delays incurred by the state length counters U3-U5 so that video timing information latched out of the state ROMs will synchronize with video data retrieved from the video RAM.

The actual state generator is made up of U3-U5, 2732 EPROMs U6 and U7, ROM address counters U8 and U9, inverter U1e and hex D flip-flop U10. At each state change, the time of the next state is latched into counters U3-U5. This time is measured in units of the dot clock period. At the same time the outputs for this state are latched into U10. These signals are, in order, (D0) blanking for the composite video, (D1) sync for same, (D2) fast count enable for the RAM address counters, (D3) and (D4) control signals for bus access arbitration, and (D5) reset for the RAM address counters.

The dot clock is divided by eight with counter U2 which generates a byte clock signal. This signal, after inversion with U1f is used to load a new byte of information from the video RAM into shift register U11 for subsequent output in the composite video. This sequential process only occurs when the display is not blanked. During blanking, the blanking signal is fed back (after a little delay through U13a) to U12a forcing the parallel load inputs of both the byte clock and the LS165 shift register low. This effectively prevents output from the byte clock from interfering with the dot clock's advance of the RAM address pointers via U14a and U12b. This fast advance of 80 counts occurs between lines so that the next 80 bytes of information fetched from the video RAM will correspond to the scan line two lines from the previous one (full interlace display). We go to all this trouble so that the screen memory addresses are contiguous and relate one-to-one with the visible lines of the display in spite of the fact that in each field only every other line is actually being displayed.

After a byte is loaded into U11 by the byte clock, it is shifted out at the dot clock rate during the display portion of the line. Open collector hex inverter gates U15a, U15b and U15c combine the output from the video signal, blanking and sync to create a composite video signal at the base of transistor Q1. This signal is amplified by Q1 operating in an emitter follower configuration. Resistors R5 and R6 provide approximately a 75 ohm output for the video signal. Since the risetimes of parts of the signal are quite short, this may cause ringing in the video amplifier portion of some monitors. This is seen as an extra bright band near the left side of larger images and a higher brightness for individual dots. Strangely enough, this effect is actually useful in some types of display as it causes thin line images made up of individual dots to merge together more smoothly than if it were not present. Capacitor C2 bypasses the video output to reduce the risetime and eliminate this effect if desired. A lot depends on the individual monitor used so it should be tried both ways to find the better setting.

Counters U16 and U17 provide the local RAM address during periods of time when data is being scanned and output to the video generator. Each time a new byte of data is loaded from the video RAM into the shift register, the address counters are incremented by 1 through U12b. At the end of each line, pin 6 of U12b is held low by the blanking signal and the output from U12b is controlled by the input on pin 5. This input is the fast count signal previously discussed.

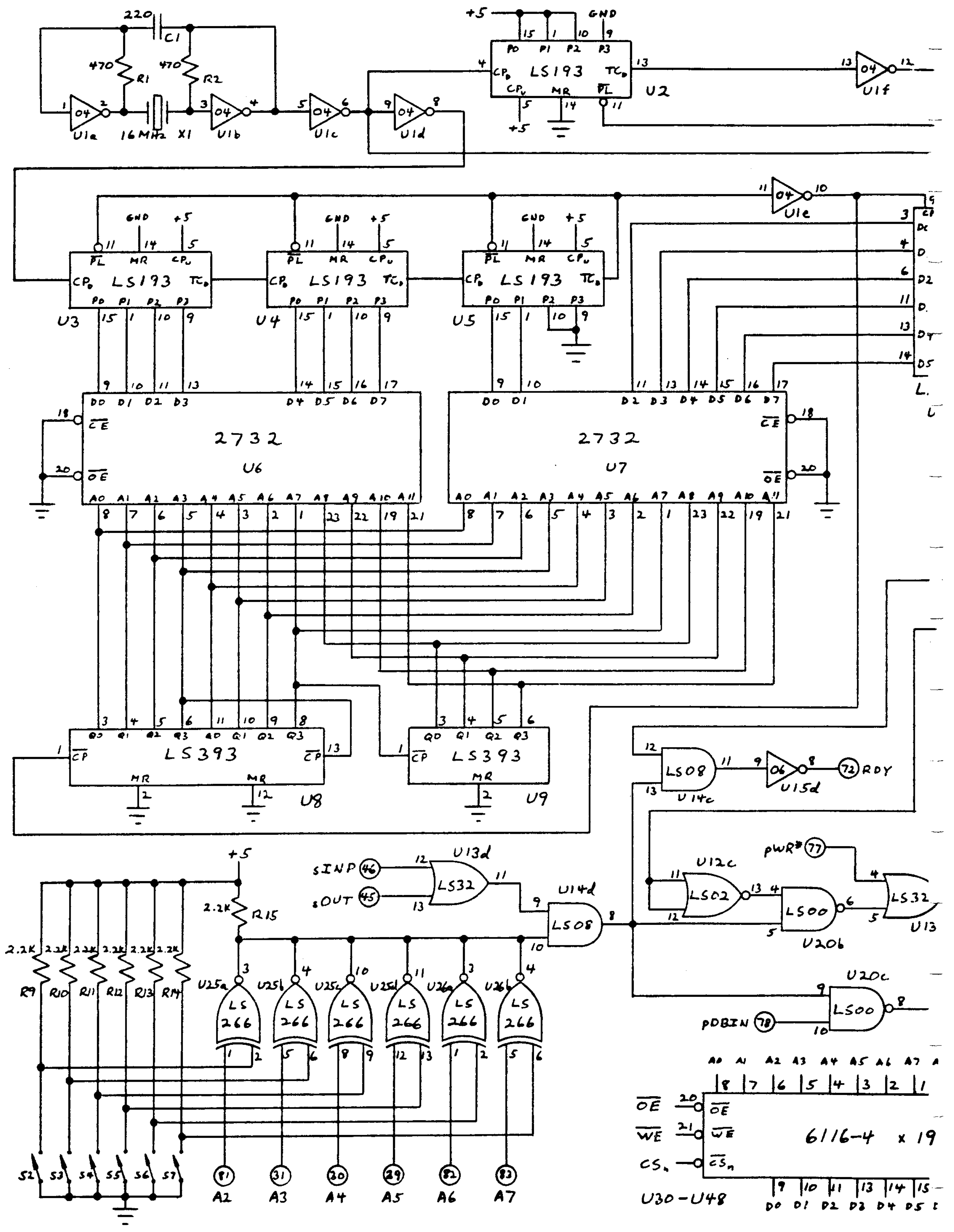
Tri-state buffers U18 and U19 buffer the local RAM address counters and drive the RAM address lines during local access. Since the video output circuit doesn't need access to the RAM during horizontal and vertical retrace, these are the times we use to allow the processor to have access to the video RAM from the S-100 bus. When the outputs from the state ROMs cause both pin 1 and pin 2 of U20a to be high, U18 and U19 are enabled and the RAM address lines are driven by the local RAM address counters. Pin 6 of U14b is just the opposite and the bus RAM address latches U21 and U22 are turned off. When either one of the inputs to U20a goes low, the outputs switch and the video RAM address lines are driven by the latches holding an address written into them from the S-100 bus. In this state the video RAM address read from or written to will be determined by the S-100 system.

Since with two bits of information we can select one of four states, you might wonder what the additional states are. To explain this, let's assume the following: say that the RAM is being accessed from the bus and a write cycle has just begun. At this moment, a state change occurs and either pin 10 or pin 12 of U11 goes high. Acting through U13b this will generate a wait state on the S-100 RDY line since pin 13 of U14c will also be high, indicating that the board is currently selected. If the processor enters a wait state before finishing the write cycle it will leave pWR* active during the entire wait state. If we allow pWR* to remain active at the RAM chips and switch to the local address counter for the scan line, a spurious write will occur on all RAM chips selected during that scan line. If we go to another state with pin 10 low and pin 12 high on U11, we maintain the wait state but in addition disconnect pWR* from the RAM chips via U12c, U20b and U13c. Once pWR* is disconnected from the RAM we can go ahead and let the local address counters select the RAM address.

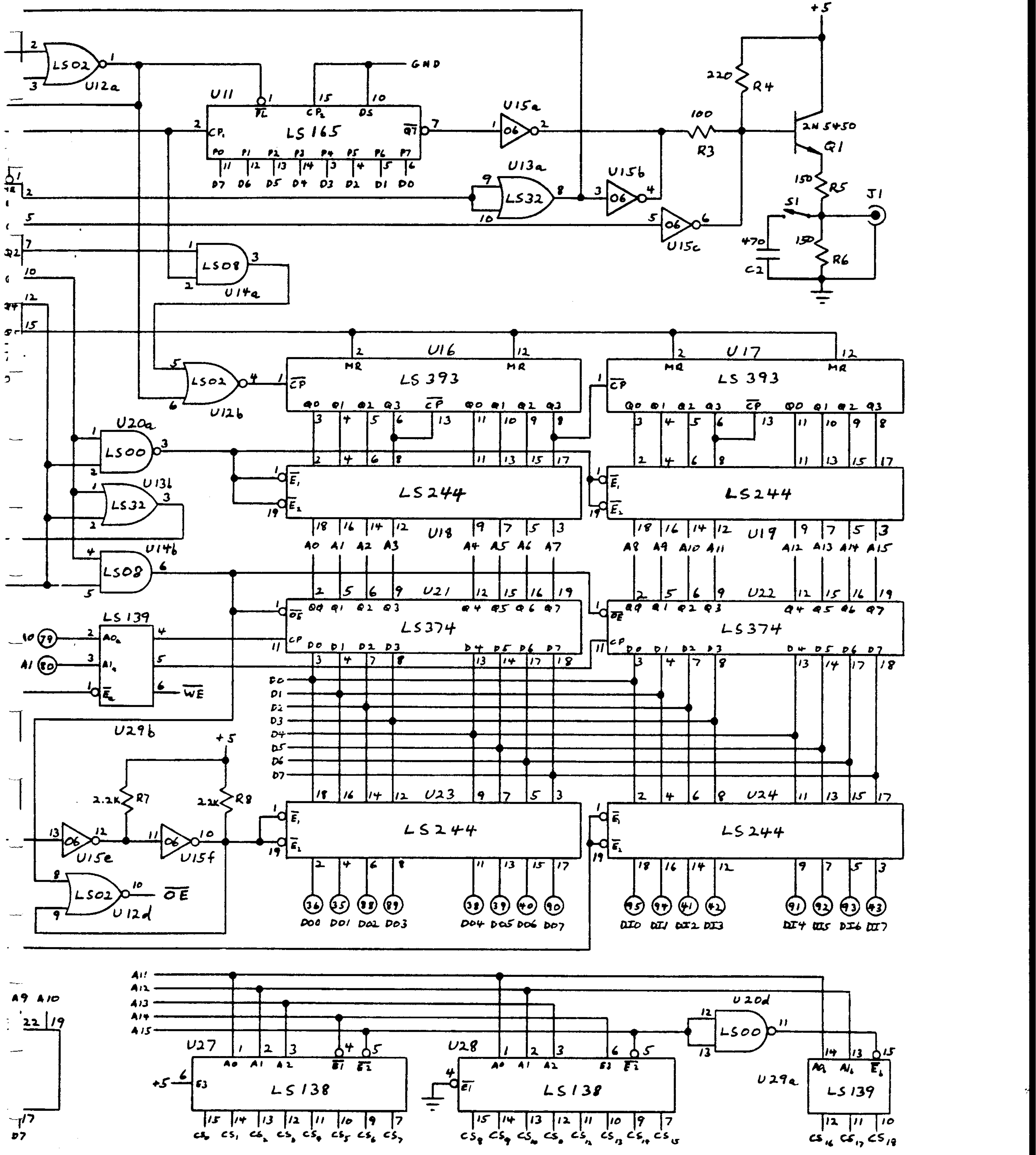
At the end of this scan line we perform the inverse operation. First we switch back over to the bus RAM address latches, and only then do we re-enable pWR*, remove the wait state and allow the processor to continue its write cycle. This arbitration scheme effectively shares the video RAM access time between the video scanning circuitry and the bus access circuitry and prevents any hash in the video display due to bus access during a scan line.

We don't have this problem during a processor read cycle because it doesn't matter if the S-100 data-in lines contain data from the RAM address currently being scanned and not the address we will ultimatley read from. The actual read operation will not complete until the desired RAM address is back on the RAM address lines and the proper data is on the data-in lines.

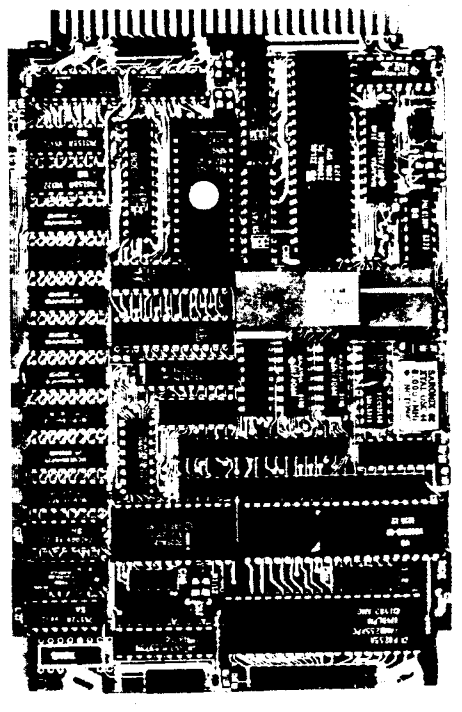
The bus access is performed in a pretty straightforward manner. Exclusive NOR gates U25 and U26a and U26b act as a comparator between the address on A2-A7 of the S-100 bus and the values preset by switches S2-S7. If the address matches and an I/O operation is selected by either sINP or sOUT high, pin 8 of U14d will go high indicating board select. If a scan line is currently in progress, a wait state will occur until pin 12 of U14c goes low. In addition pWR* will not be activated as discussed above. If the bus cycle is a read operation, pDBIN will be active and pin 8 of U20c will



NOTE: The above schematic is published for our readers' personal use. The author retains all resale rights to the design.


U12a LS02
U11 LS165
U15a 06
U13a LS32
U15b 06
U15c 06
R4 220
R3 100
2N5450 Q1
R5 150
R6 150
C2 470
S1
J1
+5
GND
U14a LS08
U12b LS02
U16 LS393
U17 LS393
U20a LS00
U13b LS32
U14b LS08
U18 LS244
U19 LS244
U21 LS374
U22 LS374
LS139
U29b
WE
R7 2.2K
R8 2.2K
U15e 06
U15f 06
U12d LS02
OE
U23 LS244
U24 LS244
DO0 DO1 DO2 DO3 DO4 DO5 DO6 DO7
DI0 DI1 DI2 DI3 DI4 DI5 DI6 DI7
A11 A12 A13 A14 A15
U27 LS138
U28 LS138
U20d LS00
U29a LS139
CS0 CS1 CS2 CS3 CS4 CS5 CS6 CS7
CS8 CS9 CS10 CS11 CS12 CS13 CS14 CS15
CS16 CS17 CS18

THE MCPU-900



(1/2 actual size)

FULL-BLOWN **8-BIT**

Microcomputer system on STD card

- Completely **STD BUS** compatible.
- 4MHz Z-80A
- 64K RAM
- 2K/4K/8K/16K/ROM or EPROM.
- Memory map under software control.
- 24 bit parallel I/O - can also serve as SASI interface for hard disk control.
- Serial I/O with 8 bits of I/O for terminal or modem control.
- Three 10MHz counter/timer channels.
- Completely programmable serial UART.
- 1797 Floppy disk controller handles four 5" or 8" disk drives. Standard drive connectors for both sizes on board. Single or double sided. Single or double density supported. All digital data separator.
- Parallel printer interface - Centronics type.
- On board interrupt handling logic.
- RAM is DMA controllable.
- MEMEX and IOEXP lines are fully implemented.
- Power-on and pushbutton reset circuitry.
- 100 hour burn-in.
- Excellent software support.

$795 Single Quantity

MILLER Technology Inc.

647 N. Santa Cruz Ave.
Los Gatos, CA 95030

(408) 395-2032

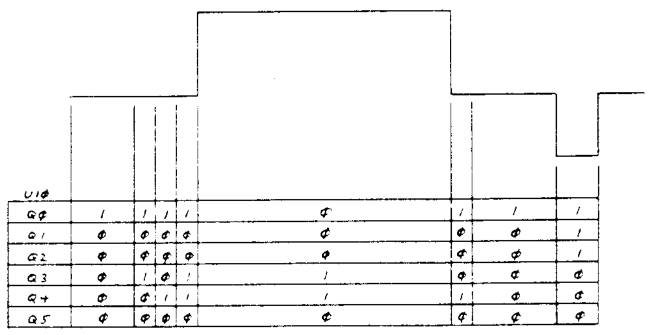
turn on the data input buffer U24. If the bus cycle is a write and bus access is allowed, the board select will combine with pWR* through U13c, U15e and U15f to enable the data output buffer U23. When this occurs, OE* to the RAM chips is turned off to prevent conflicts between a RAM chip and the data out buffer both driving the local data bus at the same time. U15e and U15f provide some delay so that the bus data out buffer holds its data valid until after WE* goes false to the 6116's. In order to select the proper device to write to, U29b is set up as an address decoder for A0-A1 of the S-100 bus. Depending on which port number of the four is selected, data will be written to the low byte bus address latch, the high byte latch or the video RAM itself at a RAM location determined by the most recent values written into the bus address latches.

Decoders U27, U28 and U29a select one of the 19 6116 RAM chips for reading or writing on both scan lines and bus accesses. U20d is used as an inverter so that U29a will select RAM address values of 8000H and up.

Though details of the arbitration may sound complicated, it is actually a pretty straightforward scheme. Figure 3 shows the state ROM outputs at the various places in a typical scan line.

Something to keep in mind here is that you need not totally understand the circuit in order to successfully build it and use it. Some experience with wire-wrapping prototypes is helpful, along with some patience since there are quite a few connections to be made, particularly in the video RAM portion of the board.

In the next part of this series I will describe how to build and check out the board, and will also provide a program for generating the state ROMs if you have access to an EPROM programmer. We can also supply preprogrammed ROMs at a nominal cost for those without such facilities. ■



U1Ø								
QØ	1	1	1	1	Ø	1	1	1
Q1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
Q2	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
Q3	Ø	1	Ø	1	1	Ø	Ø	Ø
Q4	Ø	Ø	1	1	1	1	Ø	Ø
Q5	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Figure 3: State outputs during a typical scan line.

Multi-user

A Column by E.G. Brooner

In previous columns we tried to avoid technicalities as we outlined the general kinds of milti-user systems. With all of that behind us, it's time to get a bit technical as we describe a particular kind of network (Ethernet) and a particular implementation known as "Etherseries." Very briefly, Ethernet is one of the earliest and most widespread types of network, originally meant for use with sophisticated minicomputers, and Etherseries is the particular version designed for use with the very popular IBM PC.

In contrast to many advertised net systems, this one is actually available as an off-the-shelf product and this reviewer has seen it working. Bob Metcalfe, one of Ethernet's inventors, founded the 3COM corporation to build and market network components. In his California plant we observed 50 PCs all working away at the tasks typically performed by desktop computers, all sharing one large hard disk storage system and a few strategically located printers. Figure 1 illustrates, in a simple way, how several computers are connected to form a typical network

What's Different About Ethernet?

Designers attempt to define networks within the framework of a seven-level protocol system; this sometimes leads to more confusion than clarification. Briefly, a protocol is a set of specifications for performing a certain function. If you connect a peripheral (printer etc.) to your computer, you use a certain standard method—RS232 for example. This is a "level-one" protocol, which is a definition of how two pieces of equipment communicate with one another. The way the data is bunched together for transmission to the printer constitutes a "level-two" protocol, of which there are many kinds. These two protocols, whatever they may be, constitute a complete data exchange system.

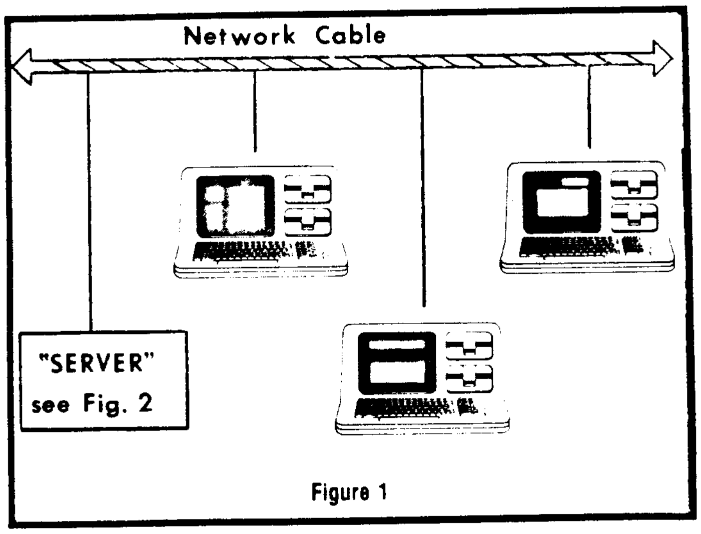


Figure 1

So much for communicating between two devices. Now, if we want to connect a larger number of devices, and selectively communicate with one or the other at various times, we need a third protocol to route and control the communication process. Three protocols define a network. The third protocol (again, whatever set of rules is used for the purpose) is what distinguishes a network from simpler systems.

Ethernet is really a definition of the two lower protocols. It defines them so clearly that any equipment connected via an Ethernet can (theoretically, at least) communicate. However, the third, or network control layer will differ as we move from one machine to another. And the level one and two protocols, which define Ethernet, are more-or-less lumped together. For example, we cannot say that it uses some particular standard or otherwise defined protocol at each level. Ethernet is a communication system that performs, in an integrated way, the functions that would otherwise require two separate protocols.

There are a lot of Ethernets (the generic term) for use with various kinds of computers. Etherseries as presently defined is only for the IBM PC. (3COM'S literature also calls the PC version Etherlink). A very similar Etherseries package would suffice for Apples or some other micros, but it would not be identical at the third level. The third level, then, is usually unique to a particular machine or operating system; it is a special, customized interface, usually combining hardware and software components. Etherseries for the PC consists of a single plug-in board and a 5" diskette and, of course, the interconnecting cable. An important point to remember is that to the user, this network appears simply as an enhancement to PCDOS, in the form of a few new commands.

How It Works.

All Ethernets have in common the CSMA/CD (carrier, sensing, multiple access, collision detection) scheme for "directing traffic" on the single coaxial cable that links all of the users. CSMA/CD is one of the two major schemes that are in use for this purpose (the other major method and the many lesser ideas for accomplishing the same thing will be discussed in future issues). When a user attempts to communicate with another device, his network software composes a "message" which includes both the data and the routing instructions. The message may be so long that it has to be broken into several smaller "packets" that will be reassembled at the receiving end of the circuit. All of this is done automatically; the user only enters simple commands, such as would be used with one of his own peripherals.

Assume that a message is ready to go. The net communication system is always "listening" to the cable. If it has not heard any signals (data) being passed for the last nine microseconds, it goes ahead and transmits the message or packet. While transmitting (remember, at something like 10 megabits per second), it continues to listen. If what it hears does not exactly match what it is sending, it assumes that either: a) an error has been made in transmission, or b) someone else has tried to transmit at the same time. In any event, it has detected a "collision" on the cable. By means of a rather complicated formula, known as the back-off algorithm, it calculates a brief waiting period and tries again. The back-off algorithm operates in such a way, and the transmissions are so rapid, that if two messages do collide on the first try, their next attempts will probably not coincide, and both will succeed.

Every station on the cable "hears" every transmission, but ignores those not addressed to it. This is typical of most networks; the undesired messages simply go on by without interrupting normal operation of the other users. In fact, a user has no way of knowing whether messages are flitting along the cable or not, unless one is addressed directly to his equipment. Even if it is, the interruption is very brief; network communication transactions typically take only fractions of a second.

As It Is Really Used

As many as 1000 PCs could conceivably be connected together, exchanging data and messages and sharing peripherals; but let's be reasonable and stick to the 50 that 3COM was using when I visited their headquarters in Mountain View, CA. First of all, each of the 50 PCs could, and usually did, operate just as if it were the only one in the building. Each had one or two floppys installed and some had printers attached; some were doing engineering work, some were for bookkeeping, and so on. At this point there was no obvious difference between this installation and any other large group of personal computers, except for the coaxial cable running unobtrusively past the rear of each machine.

But somewhere in the system there was a "disk server"—Ethershare is the trademarked name for 3COM's; almost every network has something similar. Ethershare is itself a computer complete with a keyboard, screen, gobs of memory, and a 40 megabyte hard disk. The hard disk is accessible in "chunks," each containing about the same amount of storage as a double density floppy. Figure 2 is a schematic representation of a typical server.

These subdivisions of the disk are known as "volumes." Some are accessible to one user, some to another, and some are available to anyone wishing to read them. The public volumes are the key to many of the net's features. One or more may be a common data base, or contain commonly used software, available to everyone. One or more may be used as a repository for "electronic mail" which is really a set of electronic in-baskets and out-baskets where memos can be exchanged. The others may be assigned to individual users and can be password-protected.

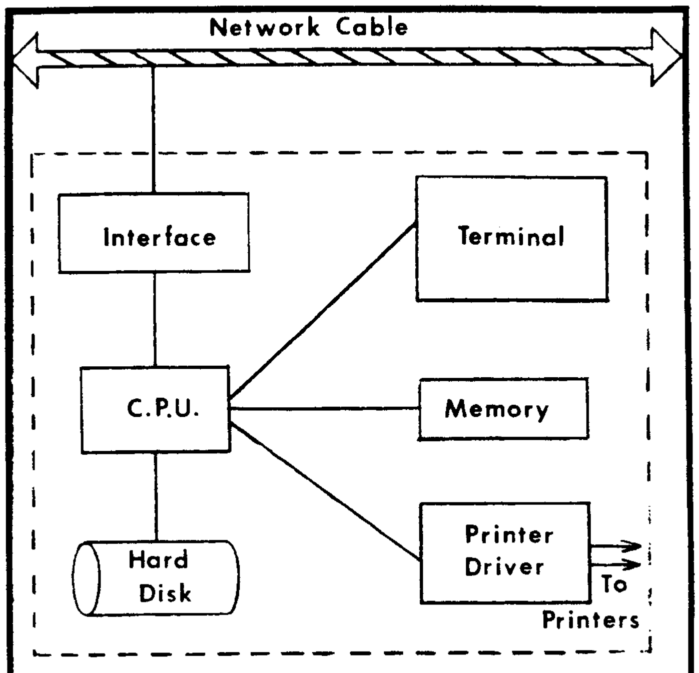


Figure 2

Say that a user has his own floppys, designated "A" and "B." He can also have two volumes on the server, designated "C" and "D." To access one of the extra storage units, he enters a simple LINK command that names the volume and equates it with "C." From that point on, until he "UNLINKs" it, he is accessing that volume, via the network, just as if it were a third drive on his own PC.

Ethershare also controls a pair of printers. Everything sent to one of the printers, from any location, is first "spooled" and then printed when the printer is free. Of course, this also frees the user to do other work after sending a print command. (See the description of print spooling which appeared in recent issues of *The Computer Journal*)

When we reviewed the operation of this particular installation it was new and the users were still in the process of adapting themselves to it. The electronic mail feature was the one which seemed to most impress those to whom it was available. The mail program included a fairly competent word-processing capability. With it a user can compose a note or letter, edit it as with any other word processor, then deposit it in one of the mail files which is accessible to the addressee. Such "mail" can be sent to any station or individual within the bounds of the network, or to a group of addressees. The mail can be accessed on-screen by the addressee anytime after it has been "sent." After that, it can be printed and/or destroyed. All messages are date and time stamped and listed on a directory. This system should be more practical than hand written memos, and it is certainly more convenient to use.

Evaluation

Etherseries for the IBM PC is an available product and not just someone's hope for the future. It is typical in many ways of any other Ethernet, and bears some resemblance to almost any of its many competitors. As far as price is concerned, it is neither the lowest nor the highest priced system available; at $950 per station, along with the relatively high initial cost of the PC, (and the server) it is probably out of the price range of all but the most serious micro users.

Its performance is superior to many competing products; the mail feature in particular is slicker than some others we reviewed. 3COM is a leader in the Ethernet business and a 3COM product, associated with an IBM product is sure to be dependable and supportable. If it is within your price range, you could do much worse than tie a bunch of IBM PCs together with this network. ■

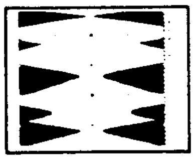
For further information, see your IBM PC dealer, or contact 3COM Corporation, 1390 Shorebird Way, Mountian View, CA, 94043.

Acknowledgement

An acknowledgement which should have been included in Volume II, Number 2 of *The Computer Journal* was inadvertently omitted. The article "Controlling DC Motors with a Microcomputer" by N. Bungard was in part made possible by research funded by National Science Foundation grant MCS-8210194.

VERSATILE DATA REDUCTION, DISPLAY AND PLOTTING SOFTWARE FOR YOUR APPLE* II

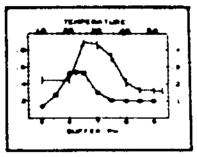
STRIPCHARTER — Turns your APPLE and Epson MX series printer into an economical *4-pen chart recorder*. Prints and displays continuous 1 to 4-channel strip-charts of any length. Ideal for large data sets. Numerous user-selectable graphics options enhance output quality. Includes 5 demos on disk with 37-page manual **$100**



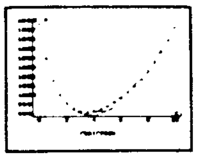
VIDICHART — Proven tool for lab data management. *Fast* plots of 4 data sets with *scrolling* in 4 directions. *zoom* scaling on X and Y axes. 2 types of graphic cursors and on-screen STATUS REPORT. even plots A D input while sampling. ADD. SUBTRACT. MULTIPLY. DIVIDE. INTEGRATE. DIFFERENTIATE. AVERAGE or NORMALIZE data sets with SIMPLE COMMANDS. Ideal for spectra, chromatograms, rate curves, etc. Includes SAMPLE DATA on disk with 28-page manual **$75**

SCIENTIFIC PLOTTER — Draws professional-looking graphs of your data. You choose data format, length and position of axes. 20 symbols, error bars, labels anywhere in 4 orientations. Includes 5 demos on disk plus 30-page manual **$25**

(For DIF file and Houston Instrument or H-P 7470A plotter adaptations, add $25 for each option selected.)



CURVE FITTER — Select the best curve to fit your data. Scale, transform, average, smooth, interpolate (3 types). LEAST SQUARES fit (3 types). Evaluate unknowns from fitted curve. Includes 5 demos on disk with 33-page manual **$35**



SPECIAL: VIDICHART SCIENTIFIC PLOTTER CURVE FITTER on 1 disk **$120**

Add $1.50 shipping on all U.S. orders. VISA or MASTERCARD orders accepted.

*Trademark of Apple Computer Inc.

INTERACTIVE MICROWARE, INC.
P.O. Box 139, Dept. 226, State College, PA 16804
CALL (814) 238-8294 for IMMEDIATE ACTION

Computer Nostalgia:

Who Invented the Personal Computer?

One of the present manufacturers likes to advertise that they invented the personal computer; I think 1978 is the year this supposedly happened. True? Absolutely not. In 1978 Apple was just beginning to be heard of; the Radio Shack model 1 was selling well, and the Commodore Pet, though less agressively marketed, had been around longer than either and was (in many opinions) a better machine.

Before any of those "personal computers" were even a gleam in their designer's eye, there were a large number of what were then called "hobby computers." The hobby computers were usually available optionally as kits or assembled products. The best known of these were the Imsai and Altair. And before the commercially available hobby computers, there were the real pioneers, individuals and groups building essentially the same thing strictly from scratch. These were the people who really "invented the personal computer."

The leader of that movement was Dr. Johnathon Titus of Blacksburg, VA. The 8008 (the first 8-bit microprocessor chip) came out in 1972. It was designed basically for control applications, but Dr. Titus, who was then doing scientific work with minicomputers, thought it had potential as the basis of a homemade personal machine.

In 1972 he managed to obtain some 8008 chips (then $120 each) and designed and built the first so-called "Mark-8" microcomputer. According to Titus, this first micro had 750 bytes (that's less than 1K) of memory! A refined, printed circuit version was written up in Radio Electronics magazine in 1974, and this marked the beginning of the movement toward hobby (or personal) computers. User groups all over the country, but primarily in southern California, built and used Mark-8 microcomputers. And in what was later to be known as "Silicon Valley" one of the earliest computer clubs, known as the Homebrew Club, saw hobbyists and professional engineers pooling their knowledge to develop even more "hobby computers." As someone said at the time, "These guys build computers in their garages and become millionaires."

By late 1974 the 8080 chip, successor to the cruder 8008, had dropped to $200 each. MITS (a now extinct company) built a kit computer around the 8080 and what is now known as the S-100 or IEEE-696 bus. They were swamped with orders before 1975 got under way. MITS' Altair was closely followed by improved look-alikes such as the (now also extinct) IMSAI and several others.

Who invented the personal computer? It certainly wasn't Apple, or even Tandy or Commodore and it happened long before 1978. *by E.G. Brooner* ■

The Computerist's Calendar

MAY 07C0

SUN	MON	TUES	WED	THURS	FRI	SAT
		1	2	3	4	5
6	7	8	9	A	B	C
D	E	F	10	11	12	13
14	15	16	17	18	19	1A
1B	1C	1D	1E	1F		

SYSTEM INTEGRATION

Part Two: Disk Controllers and CP/M 2.2 System Generation

by Bill Kibler

Disc Controllers

In part one of this series I pointed out that a system could be built with used parts for less than $1000. With this in mind, I will try to show why buying a new disk controller instead of a used one might be advisable. If you go to swap meets, you will find many of the same type of disk controllers for sale—mainly the CCS 2422. I have considerable experience with this card, and I feel that it is not a good buy. The documentation is good, but the unit's ability to interface (hardware) with other types of CPUs and memory is rather poor. The CCS 2422 is not IEEE-696 (S-100) compatible and was designed for their own cards only. These cards abound at swap meets because their owners have been unable to make them work satisfactorily.

Most CPUs, memory, and I/O ports, will work with each other, and if not compatible can usually be altered. Timing problems with I/O ports are rare, and memory is cheaper to replace than to modify. Disk controllers are another matter, however since their timing is quite critical. The VLS controllers used in the newer units are either NEC upd765 or WD 179x. Both of these have a lot of special features, some good and some bad. Common to both is the need to read the data as fast as the chip provides it. The CPU's memory fetch and write cycles must be faster than the controller's read or write times in order to clear the controller's data buffer and be ready for the next byte of data. When the timing is not correct, the unit will error and stop the data transfer. Software has advanced somewhat since the units were first made and buying a used card with no hope of upgrading is not desirable. CP/M 3.0 uses multiple banks, and consequently disk cards that are memory mapped may not work under banked operation. Some manufacturers are supplying new BIOSs and possible upgrades; these are always preferred but seldom seen at discount prices. The cost of a system without a controller is $300 to $400, and adding a new controller for $400 (with software) will still keep the system price under $1000. This new unit will be IEEE-696 compatible and therefore will work without hardware modifications. The software will be current, possibly even CP/M plus, and the utilities will aid in bringing it up. These are the premises under which I bought the the SDSystem's Versa Floppy II/696 and CP/M 3.0. However, much of what I will review is also applicable to buying used controllers or upgrading existing systems.

System Configuration

My system consists of the Computime CPU, SD's VFII, and SD's Econoram II (a 256 K banked memory). Although this is not an ideal system, it will help to illustrate the ease and the problems of system integration. The first step in any integration is understanding the individual components, their memory map, and any special problems to be handled.

The CPU is a Z80 with a serial and parallel port. The boot jump PROM or monitor will be on this card along with the serial and parallel ports. The SD memory is 256K, with one bank of 64K and four banks of 48K. One port is used to switch the banks on the memory card, port FF hex. The disk controller uses a WD 1795 and six port addresses; those switchable ports are 60 through 67 hex. The software purchased with the unit is CP/M 3.0 for their SBC-300. The sale price, including CP/M + was just under $400 (CP/M for $90!) and thus falls into our under $1000 system cost. Keeping in mind the important points covered in part one, let's see how these units rate.

a) **Documentation:** Computime is better than SDSystems.
b) **Hardware:** generally good design (both IEEE/696).
c) **Software:** SD's needs more and better documentation.
d) **Adaptability:** hardware ok, software needs help.
e) **Support:** both ok, SD needs better follow-through.
f) **Repairability:** no PROMS or PALS.

From the above quick guide, you can see that some plusses and minuses do exist. Let's review three of these points in more detail.

Documentation

Computime produces a rather complete manual on their CPU, and SDSystems could learn something by looking at it. Although I would not consider the CPU manual to be the best, it does cover the topics completely enough to make bringing up the unit fairly easy and straightforward. SD's manuals are quite brief and do not include a theory of operations. There is a section called "Functional Description" which replaces theory with an overview that hides all important facts from the user. This manual should be read carefully both for errors and for oversimplification of explanations. A case in point is the section called "Port Usage Data"; not mentioned is the fact that some entries must be complimented first (drive select). The drive select data has one line's data already complimented (bit 6, the single/double density flag), showing how confusing inadequate explanations can be even for the writers of the manual.

The listing provided for the controller is only part of the monitor and will not work as printed. SD has their own operating system (COSMOS) and the DDBIOS listing has routines for it. The disk select routines will fail if used as written. This error was discovered when I mistyped the program and the routines worked. I later discovered the typo, corrected it and found the monitor would not work. The next major problem is the lack of information on how the system was intended to interface with other components, both theirs and others. Normally, a theory of operation would describe in detail the various handshake operations, the this-before-that stuff, and would help the software hacker to write his own programs. SD must consider all information to be trade secrets, as they provide little insight into the inner workings of their cards.

Software

SDSystem's implementation of CP/M 3.0 does not have much competition to compare it with, and therefore it is hard to know how good or bad it really is. It appears that some work has gone into setting it up, but a considerable number of programs were not included when I first started on this project. SD was not supplying the full CP/M BDOSs; you got either the banked or nonbanked version, but not both. After three months of fighting, and a talk with the VP of Sales, they should now be supplying all the original Digital Research files. For the integrator this leaves only the how and why of their disk controller's timing in question. A new DDBIOS is available (they are aware of the many errors in the printed DDBIOS) but I have yet to receive mine. This condition is to be expected, and a lot of playing around will be needed to find the right software handshakes for your system.

Support and Hardware

The Versa Floppy II is the only card that I have been able to bring up without a lot of cutting and hacking. I feel that in terms of hardware, the product is quite sound and should give a lot of trouble-free service. The design is rather straightforward and, except for not having a PROM on board, I have not been able to find anything to complain about. The factory support has been about as expected, with one surprise; the support person is still there after six months. The company appears to be serious in wanting to produce a good product and improve their image, but as yet they haven't achieved that goal. They welcome constructive comments and will change their policy if a strong enough case is presented.

Making It Work

Now that we have some idea of the product and the support, let's look at what it takes to make it work. In considering how to set up the system, I toiled long over the final memory map, much as you should before starting. To make this work for most people, the system will have to come up in stages, first a monitor, then CP/M 2.2, CP/M 3.0 non-banked, and lastly CP/M 3.0 banked. Another consideration is the use of existing 2.2 systems, both SD and others. Most systems are port addressed disk controllers with some form of boot/monitor PROM (either mapped or phantom). The SD system was designed with a monitor at E800hex, and disk PROM at F000hex. Through banked switching, the loss of the memory space was minimized, but for most users this will not be an acceptable solution. The entry points to both the monitor and disk functions, however, were the F000hex entries. In my design, a fixed 2K EPROM resides at F000hex that provides both monitor and disk functions. I feel that this is fairly close to what most users will have. This design also leaves the memory open above the PROM for non-banked buffers, disk byte storage tables, SCBs, RAMdrives, or whatever. The BIOS also becomes quite short by making calls to the PROM for disk I/O, CON I/O, and initialization. My personal preference is to boot from a monitor after I know the basic system is running rather than to wonder why the disk keeps on running without anything happening. Checking the memory map listing will indicate options and give comparisons to other systems.

CP/M 2.2

Assuming that you are bringing your system up from scratch, you will have to burn a PROM for F000hex. This PROM should contain a monitor and the disk routines found in DDBIOS. Listing 1 contains the needed changes to make it run. Those routines dealing with the FORMAT portion will not fit if you include a monitor. They can be included in a separate new format program or in the PROM if an auto-boot operation is intended. I have made a separate format program because of the monitor and because I have changed the disk parameters, the number of drives and the layout. As the user of the system, you must at this point make some decisions as to the final operation of the system. Now is the time to determine the number of drives, the number of formats to be used, the types of I/O, and any other special functions you may desire. To make it easier to get the 2.2 software up and running, I have chosen to include only information for 8" single density disk drives. For transferring information and working between other systems, this is the preferred format. Larger densities require disk buffers and deblocking algorithms to combine CP/M's 128 byte sectors with the disk's actual physical sector sizes. The use of the non-standard 128 byte double density sector saves this blocking problem and is what SD used at first (users could still implement this format if needed in 2.2; 3.0 does the deblocking internally). The DDBIOS listing is somewhat mixed up when it comes to double density and will not work correctly as listed in the manual. For an easy way out, SIG/M has a ready BIOS on their disk #26, complete with a format program. The software listings provided here show how short a BIOS can be when using PROM based functions, and what is needed if only the listings from a manual are used. I recommend **The Programmer's CP/M Handbook** by Andy Johnson-Laird for more in-depth discussions on CP/M 2.2's inner workings.

For people bringing up other systems, the source code needed for the monitor is usually found in the disk controller manual, either in a monitor or as a separate BIOS program. The major advantage of bringing the unit up under CP/M 2.2 is the readily available support currently at hand. I obtained the original 2.2 BIOS from a fellow club member. This BIOS, with some modifications, was up and running in 30 minutes. This made me sure my system worked and gave me a system with which to write the BIOS for CP/M 3.0. Let's now look more closely at the necessary steps in bringing up

CP/M 2.2.

Step 1: List all ports and memory locations that will be used by the various cards. In CP/M 3.0 these are listed in PORTS. LIB. Starting a list now will make things easier later. I chose to put my disk buffers and storage locations above the PROM so that they would be protected during bank moves. When listing ports or memory locations, it helps to show what happens when accessing these entries. Note what bits contain information and how it is used. Remember that these ports/memory locations must be the same as those used in the monitor (the listings addresses are the same as DDBIOS and not above the PROM as I had suggested).

Step 2: Photocopy the listings needed for each port function. Typically these are the CON IN/OUT routines that are found in the manuals. Past experience has shown that it is best to just steal the routines word for word from the manuals. It is not unusual that the mention of timing problems which require special software never find their way to the books. So steal them all. Using the copies will be faster than trying to find them again in one of the many manuals, and also allows you to make lots of notes on the listings.

Step 3: Create the monitor program, composed of the new (stolen) I/O routines, system initialization routines (also stolen), disk functions (yes—stolen), and monitor functions (try stealing them from SIG/M disk #26). My monitor routines were originally from a CCS Z80 monitor that I converted to 8080 nemonics. Using Wordstar to block delete the old disk routines and add the new ones will speed up the operation (if the monitor is for another system). It is strongly recommended that you make small files of the routines for block adds later as these can then be turned into macros for CP/M 3.0.

Step 4: Assemble and burn the new monitor PROM. Test it to see that all functions and routines work, as CP/M will be calling these later. What you should have is a PROM at F000hex to F7FFhex with a jump table at F000hex similar to CP/M's BIOS entry table. This will make it easier to call routines, especially if you later add or delete a byte or two (the tables entry points become fixed at this time).

Step 5: Compile the new BIOS for CP/M 2.2. What will be needed are the Disk Parameter tables, any routines not in the monitor, or those that may change often. My BIOS has several of the routines which are a simple call to the PROM entry table, and a return out of it. About 600hex should be more than adequate for this type of BIOS. The CP/M systems or interface guide will list the needed routines. See the sample BIOS for more insight.

Step 6: Assemble your new BIOS, correct errors found in the assembly, and add it to a CPM60K.com file. This file is generated by "MOVCPM 60K**" or SYSGEN and doing a "SAVE 36 CPM60K.COM," when prompted to write to ? drive or reboot. This is covered in more detail in the SYSGEN manual. Use DDT to add the BIOS at 1F80hex (I like to fill 1F80 to 2500 with 00 so that dumping the memory will show if the addition is correct or not). Use "IBIOS.HEX,"cr,"R3580,"cr, then "D1F80,"cr to see if the jump table is where it is supposed to be. The "R3580" is the offset needed to load a hex file at 1F80 when it was intended to go at EA00hex.

Step 7: Save the file by "GO," then "SAVE 36 NEWCPM.COM,"cr. Use SYSGEN next and skip the "load from drive?" by hitting the return key. Write the new system to a spare disk, and then reset the system and try it. If, like myself, you are upgrading, then this testing will involve removing the old disk controller card, doing some jumper changes, a PROM change, installing the new controller, and then trying the new disk. If a second system can be borrowed for this initial start up, a lot of frustration can be saved. Do not be surprised if it does not work the first or second time. There are normally a number of typos that will need to be corrected first.

Step 8: After booting the system successfully, make lots of backups and then test it fully in all modes and ways to check for more errors. Assemble the format program and generate new disks.

For installing the new BIOS without an existing system, there are some alternatives. It is possible to generate a running system from a monitor (assuming a complete CP/M already existed for the system with only incorrect I/O codes) if the PROM can be programmed elsewhere. Some systems have the PROM/monitor on the controller along with a serial port, thus allowing them to be brought up initially from disk (the Micromation Doubler is just such a controller). Most companies have their PROMs and monitors set for their own I/O. These will need to be changed for mixed systems. To make these changes, a running monitor is needed that can do memory changes, dumps, and disk reads/writes. It may be necessary to buy such a PROM from a local dealer or fellow club member, but the cost will be low in comparison to buying all matching equipment. To change the I/O, just read in the system with the disk read function, change the I/O port addresses (must be the same length or shorter, and will have to be converted to machine language) and then rewrite to a new disk. Change the disk and try booting the system. Normally it will not work the first time, but keep trying; it will work if you have stolen all the right code.

This multiple-step method of system generation should get you up and running. Keep in mind that you will encounter plenty of obstacles, but knowing that they will appear and can be overcome should keep the frustration level low. Hopefully I have shed some light on what is needed.

Review

In this installment, I have provided some insight into the SDSystems Versa Floppy II controller, listed some things to watch for, and reviewed CPM 2.2 system generation. In the next article I will list the changes from 2.2 to 3.0 and help you generate a new non-banked BIOS.

The listings for this article are found on pages 24 and 25.

```
                       MEMORY MAP AND PORT USAGE TABLE

 STANDARD                   PROM                SDSystems  CP/M 3.0
non-prom system         based system            banked            non-banked
*FFFF      *FFFF        *FFFF      *FFFF        *FFFF                 *FFFF
 normal    extended      prom       buffers      SCB's                 SCB's
 BIOS       BIOS        monitor    RAM-DRVS     *FF00                 buffers
*F400      *F200        *F800      *F800        *F800                 *F800
 BDOS       BDOS         BIOS      monitor      monitor               monitor
*E600      *E400        *F200      *F000        *F000                 *F000
 CCP        CCP          BDOS       BIOS         resident              BIOS
*DF00      *DC00        *F400      *EA00         BIOS                 *EA00
 TPA        TPA          CCP        BDOS        *EA00                  BDOS
                        *DC00      *DC00         resident             *C800
                         TPA        CCP          BDOS                  TPA
                                   *D400        *E400
                                    TPA          TPA

                                                *C000     C000
                                                 BNK      BNK
                                                 0        1
                                                 bnk      TPA
                                                 BIOS
                                                 and
                                                 BDOS
*0100      *0100        *0100      *0100                  *0100       *0100
 CP/M       CP/M         CP/M       CP/M                   CP/M        CP/M
 pg0        pg0          pg0        pg0                    pg0         pg0
------------------------ usable TPA ------------------------------------------
 56.5K      56K          56K        54K                    58K         50K
* note the TPA of a banked CP/M 3.0 using standard BIOS (non-rom)
     is typically 60K.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
            PORT TABLE
BANK:       EQU   0FFH      ;bank switch port on SD Econoram
                            ; Disk controller ports
RESET:      EQU   060H      ;controller reset address
SELECT:     EQU   063H      ;drive select port
STATUS:     EQU   064H      ;status read port
TRACK:      EQU   065H      ;track port
SECTOR:     EQU   066H      ;sector port
DATA:       EQU   067H      ;data in/out port
CMD:        EQU   064H      ;command write port
RDCMD:      EQU   0C0H      ;read address command
RDCMD:      EQU   088H      ;read sector command
WRCMD:      EQU   0A8H      ;write sector command
WRTCMD:     EQU   0F4H      ;write track command
                            ; Comptime CPU ports
T0:         EQU   028H      ;timer #0
T1:         EQU   029H      ;timer #1
T2:         EQU   02AH      ;timer #3
TCTL:       EQU   02BH      ;timer control port
INOUT:      EQU   02CH      ;parralle1 in/out port
CONDTA:     EQU   02FH      ;serial data port
CONCTL:     EQU   02FH      ;serial control port
            EQU   02H       ;serial con in status mask
            EQU   01H       ;serial con out status mask
```

```
                  ; For the SDSystems CBIOS supplied in the
                  ; manual two areas will need changing. It is
                  ; suggested that SD CPM disk #? be obtained as
                  ; it will give you the different files already
                  ; on the disk with documentation.
                  ;
                  ; In CHGTYP: change to below code
                  ;       LDA  UNIT
                  ;       ADI  A,040H     ;MAKE ... 
                  ;       ANA  A,07FH     ;CLEAR BIT 7
                  ;       STA  UNIT
                  ;       RET
                  ;
                  ; Under the BOOT: heading remove the code
                  ;  sections BT41, BT5.,COMIN1, and HALT:
                  ; Make BT2. the below code
                  ;       CALL INITSD
                  ;       JNZ  MONITOR    ;GO TO MONITOR
                  ;                       ;DISK ERROR
                  ;   BOOTLD: ..........  ;SAME AS LISTED
                  ;                       ;IN MANUAL AND
                  ;                       ;NOW IS NEXT
                  ;                       ;AFTER BT2.
                  ;
                  ;*********************************************
                  ; CP/M BASIC INPUT/OUTPUT OPERATING SYSTEM (BIOS)
                  ; This version boots in single density
                  ; calls A,B single density.
                  ;
                  ;    syter equates and definitions start first
                  ;
F000 =            SDPROM:   EQU  0F000H     ;LOCATION OF SDPROM
003C =            MSIZE:    EQU  60         ;MEMORY SIZE IN KBYTES.
                  ;
002F =            CONDTA:   EQU  2FH        ;NEEDED TO CLEAR AT BOOT
                  ;
                  ;
                  ; sdsyter disk storage information
                  ;
0042 =            UNIT      EQU  42H        ;UNIT BYTE FOR DISK SELECT
                  ;   These and other values in the 040 to 080 hex
                  ; range could be changed to CFD4/hex range.  Some
                  ; systems start their equates with a base address
                  ; and then add to it for their byte locations.
                  ;
                  ; cpm equates
                  ;
A000 =            CBASE     EQU  (MSIZE-20)*1024  ;BIAS FOR CPM
                                                  ;LARGER THAN 20K.
D400 =            CPMB      EQU  CBASE+3400H      ;START OF CPM.
DC06 =            BDOS      EQU  CPMB+806H        ;START OF BDOS
EA00 =            BIOS      EQU  CPMB+1600H
002C =            NSECTS    EQU  44           ;NUMBER OF SECT TO LOAD
                  ;
                  ;    start of 2.2 bios
```

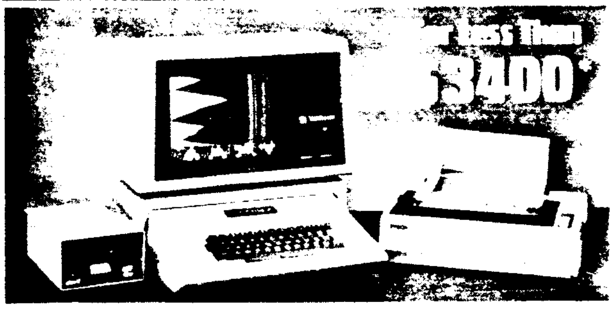
continued

```
EA00                        ORG  BIOS          ;START OF BIOS.
                  ;    entry locations for bios
                  ;    standard 2.2 entry jumps
                  ;
EA00 C37CEA                 JMP  BOOT          ;FROM COLD START LOADER.
EA03 C3D3EA      WBOOTE:    JMP  WBOOT         ;FROM WARM BOOT.
EA06 C305EB                 JMP  CONST         ;CHECK CONSOLE KB STATUS.
EA09 C309EB                 JMP  CONIN         ;READ CONSOLF CHARACTER.
EA0C C30DEB                 JMP  CONOT         ;WRITE CONSOLE CHARACTER.
EA0F C35CEB                 JMP  LIST          ;WRITE LISTING CHAR.
EA12 C362EB                 JMP  PUNCH         ;WRITE PUNCH CHAR.
FA15 C363EB                 JMP  READER        ;READ READER CHAR.
EA18 C3CEEA                 JMP  HOMF1         ;MOVE DISK TO TRACK ZERO.
EA1B C3B6EA                 JMP  TDSKSL        ;SELECT DISK DRIVE.
EA1E C31EF0      SETTRK:    JMP  SDPROM+1EH    ;SEEK TO TRACK IN REG A.
EA21 C321F0      SETSEC:    JMP  SDPROM+21H    ;SET SECTOR NUMBER.
EA24 C324F0                 JMP  SDPROM+24H    ;SET DISK STARTING ADR.
EA27 C327F0                 JMP  SDPROM+27H    ;READ SELECTED SECTOR.
EA2A C32AF0                 JMP  SDPROM+2AH    ;WRITE SELECTED SECTOR.
EA2D C360EB                 JMP  PRSTAT        ;LIST STATUS CHECK
EA30 C364EB                 JMP  SECTRAN       ;SECTOR TRANSLATE ROUTINE

                  ;              DPH
                  ;    disk parameter headers for the disk drives
                  ;
EA33 =           DPBASE: EQU     $     ;BASE OF DISK PARAMETER BLOCKS
EA33 62EA0000    DPE0:   DW      XLT0,0000H       ;TRANSLATE TABLF
EA37 00000000            DW      0000H,0000H      ;SCRATCH AREA
EA3B 6CEB53EA0B          DW      DIRBUF,DPB0      ;DIR BUFF,PARM BLOCK
                    DW      CSV0,ALV0
EA43 62FA0000    DPE1:   DW      XLT0,0000H       ;TRANSLATE TABLE
FA47 00000000            DW      0000H,0000H      ;SCRATCH AREA
FA4B 6CFB53EA            DW      DIRBUF,DPB0      ;DIR BUFF,PARM BLOCK
FA4F 3AFC1BFC            DW      CSV1,ALV1        ;CHECK, ALLOC VECTORS
                  ;
FA53 =           DPB0:   EQU     $                ;SD DISK PARM BLOCK
FA53 1A00                DW      26               ;SEC PER TRACK
FA55 03                  DB      3                ;BLOCK SHIFT
FA56 07                  DB      7                ;BLOCK MASK
FA57 00                  DB      0                ;EXTNT MASK
FA58 F200                DW      242              ;DISK SIZE-1
EA5A 3F00                DW      63               ;DIRECTORY MAX
FA5C C0                  DB      192              ;ALLOC0
FA5D 00                  DB      0                ;ALLOC1
FA5F 1000                DW      16               ;CHECK SIZE
FA60 0200                DW      2                ;OFFSET
EA62 =           XLT0:   EQU     $                ;TRANSLATE TABLE
FA62 0107D1319           DB      1,7,13,19,25,5,11,17,23
FA6B 03090F1502          DB      3,9,15,21,2,8,14,20,26
EA74 060C121804          DB      6,12,18,24,4,10,16,22
                  ;
                  ;
                  ; BOOT
                  ; This is the first entry entered from monitor
                  ; after reset and after system is loaded from
                  ; system track (tk 0 and 1)
                  ;
EA7C 318000      BOOT:    LXI  SP,80H       ;SET STACK POINTER.

                  ;    if all initlization is not done in monitor
                  ;    enter code here ******
                  ;
EA7F AF                   XRA  A
FA80 320400               STA  4            ;CDISK
EA83 320300               STA  3            ;IOBYTE
EA86 322A00               STA  42
FA89 CD9BEA               CALL SETUP        ;SET UP JUMPS.
EA8C DB2F                 IN   CONDTA       ;CLEAR CONSOLE STATUS.
EA8E 211CEB               LXI  H,OMSG       ;PRINT OPENING MESSAGE.
EA91 CD11FB               CALL PMSG
FA94 3A0400      GOCPM:   LDA  4            ;GET DISK NUMBER TO
FA97 4F                   MOV  C,A          ;PASS TO CCP IN C.
EA98 C300D4               JMP  CPMB         ;JUMP TO CCP.
                  ;
                  ; SET UP JUMPS TO CP/M
                  ;
EA9B 3EC3        SETUP:   MVI  A,0C3h       ;PUT JMP TO WBOOT
FA9D 320000               STA  0            ;ADR AT ZERO.
EAA0 2103EA               LXI  H,WBOOTE
EAA3 220100               SHLD 1
EAA6 320500               STA  5
EAA9 2106DC               LXI  H,BDOS       ;PUT JUMP TO BDOS
EAAC 220600               SHLD 6            ;AT ADR 5,6,7.
EAAF 2180002240           LXI  H,80H        ;SET DEFAULT DMA ADR.
                     SHLD 40h
EAB5 C9                   RET               ;RETURN FROM SETUP.
                  ;
                  ; The following storeges the SD code for the drive
                  ; in the unit memory location. A&B are single density
                  ; if more drives needed change code here and add
                  ; more DPH's- one per drive.
                  ;
EAB6 210000      TDSKSL:  LXI H,0
FAB9 79                   MOV A,C
EABA FE02                 CPI 2             ;FIND IF MORE THAN TWO DISKS
EABC D0                   RNC               ;SELECTED IF SO RETURN
EABD E6FD                 ANI 0FDH          ;REMOVE THEN CPM BITS
FABF 324200               STA UNIT          ;SET THE DISK UNIT MEMORY LOCAT
EAC2 69                   MOV L,C           ;NOW SET UP THE HL REGS FOR
FAC3 2600                 MVI H,0           ;LOCATION OF THE PARAMETERS
EAC5 1133EA               LXI D,DPBASE      ;FOR THE SELECTED DRIVE
FAC8 29                   DAD H
FAC9 29                   DAD H
EACA 29                   DAD H
EACB 29                   DAD H
EACC 19                   DAD D
EACD C9                   RET
                  ;
                 HOMF1:
EACF 0F00                 MVI      C,0
EAD0 C31EEA               JMP      SETTRK


                  ;
                  ; WARM-BOOT:  READ ALL OF CPM BACK IN
                  ; EXCEPT BIOS, THEN JUMP TO CCP.
                  ;
EAD3 318000      WBOOT:   LXI  SP,80H       ;SET STACK POINTER.
EAD6 3A4200               LDA  42H          ;SAVE DISK NUMBER.
FAD9 326BEB               STA  TEMP
```

```
EADC 3E00                 MVI  A,0          ;0 for single and 40 for
EADF 324200               STA  42H          ;DOUBLE
EAE1 0E00                 MVI     C,0
EAE3 CD1EFA               CALL    SETTRK
EAE6 3E2C                 MVI A ,NSECTS     ;GET # SECTORS FOR CPM READ.
EAE8 324500               STA  45H
EAEB 0E02                 MVI C,2
EAED CD21FA               CALL  SETSEC
EAF0 2100D4               LXI  H,CPMB       ;GET STARTING ADDRESS.
EAF3 224000               SHLD  40H
EAF6 CD2DF0               CALL SDPROM+2DH
EAF9 3A6BFB               LDA      TEMP
EAFC 324200               STA  42H
EAFF CD9BFA               CALL SETUP        ;SET UP JUMPS.
EB02 C394FA               JMP  GOCPM        ;GO BACK TO CPM.
                  ;
                  ;
                  ; CHECK CONSOLE INPUT STATUS.
                  ;

EB05 CD06F0C9  CONST:  CALL   SDPROM+06H    ;READ CONSOLE STATUS.
              RET              ;RETURN FROM CONST.

                  ;
                  ; READ A CHARACTER FROM CONSOLE.
                  ;
EB09 CD09F0    CONIN:     CALL SDPROM+09H
EB0C C9                   RET
                  ;
                  ; WRITE A CHARACTER TO THE CONSOLE DEVICE.
                  ;

EB0D CD0CF0    CONOT:  CALL   SDPROM+0CH
EB10 C9                   RET



                  ; PRINT THE MESSAGE AT H&L UNTIL A ZERO.
                  ;
EB11 7E        PMSG:     MOV  A,M         ;GET A CHARACTER.
EB12 B7                  ORA  A           ;IF IT'S ZERO,
EB13 C8                  RZ               ;RETURN.
EB14 4F                  MOV  C,A         ;OTHERWISE,
EB15 CD0DFB              CALL CONOT       ;PRINT IT.
EB18 23                  INX  H           ;INCREMENT H&L,
EB19 C311FB              JMP  PMSG        ;AND GET ANOTHER.
                  ;
                  ; CBIOS MESSAGES
                  ;


EB1C 0D0A43502FOMSG:   DB   0DH,0AH,'CP/M 2.2 SD SYSTEMS-KIBLER  '
EB3A 0D0A392253        DB   0DH,0AH,'8"SD A&B '
EB45 0D0A3630          DB   0DH,0AH, MSIZE/10+'0',MSIZE MOD 10 + '0'
EB49 4B20563125        DB   'K V1.0 of 2/01/84 ',0
                  ;
                  ; WRITE A CHARACTER ON LIST DEVICE.
                  ; INSERT YOUR ROUTINE HERE
EB5C CD12F0    LIST:   CALL   SDPROM+12H
EB5F C9                RET
                  ;
EB60 AF        PRSTAT  XRA A
EB61 C9                RET      ;RETURN ALWAYS NOT READY
                  ; PUNCH PAPER TAPE.
                  ;
               PUNCH:
EB62 C9                RET                  ;RETURN FROM PUNCH.
                  ;
                  ;  NORMALLY USED TO READ PAPER TAPE.
                  ;
               READER:
EB63 C9                RET                  ;RETURN FROM READER.

                  ;SECTOR TRANSLATION ROUTINE FOLLOWS
EB64 EB        SECTRAN XCHG
EB65 096E              DAD     B
              MOV L,M
EB67 2600              MVI H,0
EB69 C9                RET

EB6A            TEMP:   DS    1
                  ;
                  ; DISK DATA STORAGE AREA DONOT CHANGE
EB6B =          BEGDAT  EQU      $
EB6B            DIRBUF: DS       128      ;DIRECTORY ACCESS BUFFER
EBEB            ALV0:   DS       31
EC0A            CSV0:   DS       16
EC1A            ALV1:   DS       31
EC39            CSV1:   DS       16
EC49 =          ENDDAT  EQU      $
00DD =          DATSIZ  EQU      $-BEGDAT
                  ;
EC49            END
```

```
                   ;     BOOT LOADER
                   ;
                   ;this is a quick bootstrap that loads at track 0
                   ;sector 1 it will be put into memory at 0000 by the
                   ;Sdprom bootstrap disk read then it will be
                   ;executed and read in the rest of the first two tracks
                   ;
003C =             MSIZE   EQU     60
F000 =             SDPROM  EQU     0F000H
EA00 =             BIOS    EQU     5A00H+(MSIZE-24)*1024
0000 318000                LXI     SP,80H
0003 210200                LXI     H,2H            ;SET TO 2ND SECTOR
0006 224300                SHLD    43H             ;SECTOR TO READ
0009 2100D4                LXI     H,BIOS-1600H    ;START OF CPM'S CCP
000C 224000                SHLD    40H             ;DMA LOCATION
000F 3E33                  MVI     A,51
0011 324500                STA     45H             ;NUMER OF SECTORS TO READ
0014 CD2DF0                CALL    SDPROM+2DH      ;READ MULTIPLE SECTORS
0017 C300EA                JMP     BIOS
001A                       END
```
■

ADALAB™ Automates Lab Instruments



- Interactive Microware's general-purpose ADALAB™ data acquisition and control system interfaces with virtually any lab instrument using a recorder or meter, including GC and HPLC systems, spectrophotometers, pH meters, process control apparatus, thermocouples, etc.
- Lab Data Manager™ software facilitates single or multi-channel acquisition, storage, display and chart recorder style output of lab instrument data. IMI QUICKI/O software operates within easy-to-use BASIC!
- Thousands of scientists currently use IMI software and/or ADALAB products worldwide!

*Price includes 48K APPLE† II+ CPU, disk drive with controller, 12" monitor, dot matrix printer with interface, IMI ADALAB™ interface card.

†Trademark of Apple Computer Inc

Attention APPLE Owners

IMI's ADALAB INTERFACE CARD IS AVAILABLE SEPARATELY FOR ONLY $495

(Includes 12-bit A/D, 12-bit D/A, 8 digital sense inputs, 8 digital control outputs, 32-bit real-time clock, two 16-bit timers plus QUICKI/O data acquisition software.)

imi

INTERACTIVE MICROWARE, INC.
P.O. Box 771, Dept. 226
State College, PA 16801 (814) 238-8294

Editor's Page, continued from page 1

class bulk mailing.

It is obvious from looking at these programs that they were designed by a programmer who had no idea of what goes on in the real world. There probably *are* some useful programs available, but the ones we looked at did not fit our needs at all. Fortunately Ernie Brooner, who is writing our multi-user column, offered a data base which he wrote for his own use. It is not exactly what we need, but Ernie said "That's simple—I wrote it, so I'll just change a few lines."

The difference between a program written by a user and one written by a programmer who does not actually use the software himself is obvious to anyone who tries to use the program in a day-to-day basis.

It is time for the software industry to get out in the field and spend some time with their customers to learn what it is that people really *do* with computer software. ■

The Bookshelf

TTL Cookbook
Popular Sams author Dan Lancaster gives you a complete look at TTL logic circuits, the most inexpensive, most widely applicable form of electronic logic. In no-nonsense language, he spells out just what TTL is, how it works, and how you can use it. Many practical TTL applications are examined, including digital counters, electronic stopwatches, digital voltmeters, and digital tachometers. By Don Lancaster. 336 pages, 5½x8½, soft. © 1974 .... $11.95

SCRs and Related Thyristor Devices
A comprehensive guidebook to the operational theory and practical applications for silicon controlled rectifiers, triacs, diacs, unijunction transistors, and other members of the thyristor family. Also contains a microprocessor mini-course to help you in inter-facing thyristors with digital control circuits. If you're involved with design, installation, or maintenance of electronic power-control equipment, this is the book for you. By Clay Laster. 136 pages, 8½x11½, soft. © 1981 .... $12.95

Instrumentation: Transducers, Experimentation, and Applications
A laboratory-oriented manual that helps provide you with an in-depth understanding of instrumentation and measurement. By Roger W. Prewitt and Stephen W. Fardo. 224 pages, 8½x11, soft. © 1979 .... $12.95

The Programmer's CP/M Handbook
An exhaustive coverage of CP/M-80®, its internal structure and major components is presented. Written for the programmer, this volume includes subroutine examples for each of the CP/M system calls and information on how to customize CP/M – complete with detailed source codes for all examples. A dozen utility programs are shown with heavily annotated C-language source codes. An invaluable and comprehensive tool for the serious programmer. By Andy Johnson-Laird. 750 pages, 7½x9¼, softbound $21.95

Interfacing to S-100 (IEEE 696) Microcomputers
This book is a must if you want to design a custom interface between an S-100 microcomputer and almost any type of peripheral device. Mechanical and electrical design is covered, along with logical and electrical relationships, bus interconnections and more. By Sol Libes and Mark Garetz. 322 pages, 6½x9¼, softbound $16.95

Microprocessors for Measurement and Control
You'll learn to design mechanical and process equipment using microprocessor-based "real time" computer systems. This book presents plans for prototype systems which allow even those unfamiliar with machine or assembly language to initiate projects. By D.M. Auslander and P. Sagues. 310 pages, 7 3/8x9 1/4, softbound $15.99

Osborne CP/M® User Guide (Second Edition)
A new revised edition which includes expanded sections on CP/M® 86 and CP/M® 80, as well as CP/M® 's relationship to assembly language programming, MP/M® and CP/NET® operating environments. By Thom Hogan. 292 pages, 6½x9¼, softbound $15.95

Discover FORTH
Whether you are a beginner seeking information on this multi-faceted programming language or a serious programmer already using FORTH, this book is a reference that should not be overlooked. Long considered a computer language of building blocks, FORTH has been optimized for speed and requires little computer support. By Thom Hogan. 146 pages, 6¼x9¼, softbound $16.95

68000 Assembly Language Programming
Each of the 68000's instructions is individually presented and fully explained in this assembly language tutorial. For experienced programmers, this book is also a complete reference to the 68000 instruction set and programming techniques. By Lance A Leventhal. 614 pages, 6½x9¼, softbound $18.95

Z8000® Assembly Language Programming
This book is filled with real-world programming examples, sample problems, and troubleshooting hints that will guide the reader to mastery of this powerful new 16-bit "super chip". The entire Z8000® instruction set is described in detail. By Lance A Leventhal, Adam Osborne, and Chuck Collins. 928 pages, 6½x9¼, softbound $19.99

The 8086 Book
Anyone using, designing, or simply interested in an 8086-based system will be delighted by this book's scope and authority. As the 16-bit microprocessor gains wider inclusion in small computers, this book becomes invaluable as a reference tool which covers the timing, architecture and design of the 8086, as well as optimal programming techniques, interfacing, special features, and more. By Russell Rector and George Alexy. 624 pages, 6½x9¼, softbound $16.99

Z80® Assembly Language Programming
Programming examples illustrate software development concepts and actual assembly language usage. More than 80 sample programming problems with solutions and a complete Z80® instruction set reference table. By Lance A. Leventhal. 640 pages, 6½x9¼, softbound $18.95

8080A/8085 Assembly Language Programming
More quality programming examples and instruction sets than can be found in any other book on the subject. Information on assemblers, program loops, code conversion and more. A must for 8080A/8085 programmers. By Lance A Leventhal. 448 pages, 6½x9¼, softbound $18.95

Microprocessor Circuits, Volume 2: I/O Interfacing & Programmable Controllers
Ideal way to learn about comercial and industrial applications of microprocessor circuitry and gain practical, valuable, hands-on experience at the same time. Features many easy-to-build demonstration circuits that teach you about advanced microprocessors, microcontrollers, and real-world I/O interfacing. Perfect for technicians, hams, students, and teachers. By Edward M. Noll. 128 pages, 8½x11, softbound $9.95

IC Timer Cookbook (2nd edition)
Learn more ways to use the IC timer in this big Second Edition of Sam's best-seller. It's easy to use, practical, and includes many new devices with ready-to-use applications in circuits that really work! All circuits and relationships are fully defined and discussed for clarity. You'll know a lot more about a lot more ICs after you've finished this one. By Walter G. Jung. 384 pages, 5½x8½, softbound $17.95

Microprocessor-Based Robotics
Introduces you to robotics – a dynamic new field of science that uses your computing and electronic talents as well as your mechanical and electrical knowledge. First, you'll learn the mechanics of robot hands, arms, and legs; then, tactile sensing, motion and attitude sensing, and vision systems. After that, you learn controlling with microprocessors and BASIC programs, and finally, you learn to control the entire robot system with voice commands! Fascinating and *not* machine specific. By Mark J. Robillard. 224 pages, 8½x11, softbound $16.95

TV Typewritter Cookbook
Shows you how to quickly and easily project words and pictures from a common, microprocessor-based system onto an ordinary TV set. You'll be introduced to TVT communications by best-selling author Don Lancaster, who discusses basic TVT system design, memory types, interface circuitry, hard-copy output, and color graphics. By Don Lancaster. 256 pages, 5½x8½, softbound $11.95

Microcomputer Math
A step-by-step introduction to binary, octal, and hexidecimal numbers, and arithmetic

operations on all types of microcomputers. Excellent for serious BASIC beginners as well as assembly-language programmers. Treats addition and subtraction of binary, multiple-precision and floating-point operations, fractions and scaling, flag bits, and more. Many practical examples and self-tests. By William Barden. 160 pages. 5½x8½, softbound $11.95

Understanding Digital Logic Circuits

A working handbook for service technicians and others who need to know more about digital electronics in radio, television, audio, or related areas of electronic troubleshooting and repair. You're given an overview of the anatomy of digital-logic diagrams and introduced to the many commercial IC packages on the market. By Robert G. Middleton. 392 pages. 5½x8½, softbound .... $18.95.

CMOS Cookbook

One of the best-selling technical books on the market, this cookbook gives you a solid understanding of CMOS technology and its application to real-world circuitry. Explains how CMOS differs from other MOS designs, how it's powered, and what its advantages are over other constructions. The final chapter shows you how to put all preceding information to work constructing several large-scale, working instruments. Includes a mini-catalog of more than 100 devices with pinouts and application notes. By Don Lancaster. 416 pages. 5½x8½, softbound $13.95

SCRs and Related Thyristor Devices

A comprehensive guidebook to the operational theory and practical applications for silicon controlled rectifiers, triacs, diacs, unijunction transistors, and other members of the thyristor family. Also contains a microprocessor mini-course to help you in interfacing thyristors with digital control circuits. If you're involved with design, installation, or maintenance of electronic power-control equipment, this is the book for you. By Clay Laster. 136 pages, 8½x11, softbound $12.95

Real Time Programming: Neglected Topics

This book presents an original approach to the terms, skills, and standard hardware devices needed to connect a computer to numerous peripheral devices. It distills technical knowledge used by hobbyists and computer scientists alike to useable, comprehensible methods. It explains such computer and electronics concepts as simple and hierarchical interrupts, ports, PIAs, timers, converters, the sampling theorem, digital filters, closed loop control systems, multiplexing, buses, communication, and distributed computer systems. By Caxton C. Foster. 190 pages, 6¼x9¼, softbound $9.95

Interfacing Microcomputers to the Real World

Here is a complete guide for using a microcomputer to computerize the home, office, or laboratory. It shows how to design and build the interfaces necessary to connect a microcomputer to real-world devices. With this book, microcomputers can be programmed to provide fast, accurate monitoring and control of virtually all electronic functions – from controlling houselights, thermostats, sensors, and switches, to operating motors, keyboards, and displays. This book is based on both the hardware and software principles of the Z80 microprocessor (found in several minicomputers, Tandy Corporation's famous TRS-80, and others). By Murray Sargent III and Richard Shoemaker. 288 pages. 6¼x9¼, softbound $15.55

IC Timer Cookbook

Gives you a look at the hundreds of ways IC timers are used in electronics. Provides a collection of numerous recipes for using the IC timer, including a 555 monostable circuit with auxiliary output, a touch switch, a programmable monostable circuit,and hundreds of others. By Walter G. Jung. 288 pages. 5½x8½, soft. © 1977 .... $10.95

CP/M Primer

Helps microcomputer veterans and novices alike find the answers about CP/M in a complete, one-stop sourcebook that's a Sams best-seller! Gives you complete CP/M terminology, hardware and software concepts, startup details, and more for this popular 8080/8085/Z-80 operating system. Helps you begin using and working with CP/M immediately, and includes a list of compatible software, too. By Stephen Murtha and Mitchell Waite. 96 pages. 8½x11, comb. © 1980 .... $14.95

Soul of CP/M: Using and Modifying CP/M's Internal Features

Teaches you how to modify BIOS, use CP/M system calls in your own programs, and more! Excellent for those who have read *CP/M Primer* or who otherwise understand CP/M's outer-layer utilities. By Mitchell Waite. Approximately 160pages, 8x9½, comb. © 1983 .... $14.95

The S-100 and Other Micro Buses (2nd Edition)

Examines microcomputer bus syestems in general and 21 of the most popular systems in particular, including the S-100. Helps you expand your computer system through a better understanding of what each bus includes and how you can interface one bus with another. By Elmer C. Poe and James C. Goodwin, II. 208 pages. 5½x8½, soft. © 1981 $9.95

Interfacing & Scientific Data Communications Experiments

This book introduces you to the principles involved in transferring data using the asynchronous serial data-transfer technique. It focuses on using the universal asynchronous receiver/transmitter (UART) chip in order to help your understanding of communication chips. Explores operation of teletype-writer interfaces and serial transmission circuits. With experiments and circuit details. By Peter R. Rony. 160 pages 5½x8½, soft. © 1979. .... $7.95

Active-Filter Cookbook

A practical discussion of the many active-filter types and uses, written by one of Sams' most popular authors. Teaches you how to construct filters of all types, including high-pass, low-pass, and bandpass having Bessel, Chebyshev, or Butterworth response. Easy to understand – no advanced math or obscure theory. Can also be used as a reference book for analysis and synthesis techniques for active-filter specialists. By Don Lancaster. 240 pages. 5½x8½, soft. © 1975 .... $14.95

IC Converter Cookbook

Discusses and explains data conversion fundamentals, hardware, and peripherals. A valuable guide to help you understand and use d/a and a/d converter applications. Includes manufacturers' data sheets. By Walter G. Jung. 576 pages. 5½x8½, soft. © 1978.... $14.95

IC Op-Amp Cookbook

An informal, easy-to-read guide covering basic op-amp theory in detail, with 200 practical, illustrated circuit applications to reflect the most recent technology. JFET and MOSFET units are shown in both single and multiple formats. Includes manufacturers' data sheets, and lists addresses of the companies whose products are featured. By Walter G Jung. 480 pages. 5½x8½, soft. © 1980 .... $15.95

Regulated Power Supplies (3rd Edition)

Newest, most comprehensive discussion you'll find of regulated power supplies, including their internal architecture and operation. Thoroughly explains how to use regulation in your designs and projects when the need arises, and discusses practical circuitry and components. A valuable book for any technician or engineer involved in servicing or design. By Irving M. Gottlieb. 424 pages. 5½x8½, soft. © 1981 .... $19.95

The Computer Journal

PO Box 1697 Kalispell, MT 59903

Order Date: ____________________

Print Name ____________________

Address ____________________

City ____________ State ________ Zip ________

☐ Check ☐ Mastercard ☐ Visa

Card No ____________ Expires ________

Signature for Charge ____________________

Qty	Title	Price	Total

Shipping charges are: $1.00 for the first book, and $.50 for all subsequent books. Please allow 4 weeks for delivery.

Book Total ________

Shipping ________

TOTAL ________

New Products

SYBEX Releases "Mastering CP/M."

Mastering CP/M, an advanced guide to using, altering, and adding features to the CP/M microcomputer operating system, has just been released by SYBEX. CP/M users and systems programmers will better understand the organization and operation of CP/M with this book. The BIOS (Basic Input/Output System) and the BDOS (Basic Disk Operating System), are described in detail, illuminating for the reader the subtleties of the useful CP/M system. Macros instructions, powerful tools that enable programmers to design more efficient assembly language programs, are introduced, and a valuable library of macros is developed.

This well-written and fascinating book takes the reader on a step-by-step journey of discovery, leading to a more thorough understanding of the organization and operation of CP/M. An important set of appendices is included, making this a comprehensive reference for CP/M users and programmers. The book is priced at $17.95. Add $1.50 for postage when ordering directly from SYBEX, 2344 Sixth Street, Berkeley, CA, 94710. ■

Free Thermistor Catalog

Thermometrics, Inc. of Edison, New Jersey announces the publication of it's 52 page Thermistor Catalog number 181-D. The new catalog will prove to be of great value to anyone who has to design, specify or use thermistors and thermistor networks. Some of the useful features are as follows.

• A four page foldout "Thermistor Selection Guide" which provides comparison of all the styles and sizes of thermistors at Thermometrics, and includes physical, thermal and electrical properties for each type.

• A review of the extensive calibration and test facilities and services available at Thermometrics.

• A technical applications and data section which includes definitions of thermistor terminology, the various equations which describe the thermistor R-vs-T characteristics, a discussion of curve tolerances and two design examples on linearized voltage and resistance networks including output "S" curves for different material systems.

• A product section for each of the standard thermistor types available detailing all dimensions, R-vs-T characteristics, thermal properties, options and ordering information.

In addition to the new catalog there are some very useful application notes available which deal with thermistor theory, measurement, design techniques, stability and theory of self heated thermistors (including their use in flow measurement.) This information is available free of charge to interested readers from Thermometrics, 808 US Highway 1, Edison, New Jersey, 08817, Tel. 201-287-2870. ■

FORTH Tutorial at Half Price

MicroMotion announces the availability of the **FORTH-79 Tutorial & Reference Manual** at half price ($10.00). This professionally written manual was the first complete FORTH tutorial to teach the FORTH computer language, including FORTH-79 and FIG-FORTH. It has been replaced with their new publication, **FORTH Tools** ($20.00), which teaches the new 1983 International Standard. For further information contact MicroMotion, 12077 Wilshire Blvd. #506, Los Angeles, CA, 90025, Tel. 213-821-4340. ■

Interface Breadboard Package from Group Technology

The Color Computer Expansion Connector Breadboard, Model CC-100, for the TRS-80 Color Computer 1 or 2 makes it possible to connect external devices to the expansion connector signals of the computer. Combined with a solderless breadboard and the book **TRS-80 Color Computer Interfacing, With Experiments** (book no. 21893), it forms the CoCo-100 package providing basic interfacing instructions for any version of this versatile computer. In addition, the CC-100 Experiment Component Package contains the parts necessary to do the experiments in the book.

With the CoCo-100, the user can learn in step-by step fashion how to access the signals available in the parallel expansion connector of the TRS-80 Color Computer and how to construct and use a peripheral interface adapter (PIA). The experiments demonstrate how to enter and retrieve binary data and how analog-to-digital and digital-to-analog conversion is performed both within the computer and using external devices. With the fundamental understanding and hands-on experience developed through the interface package, users are well-equiped to extend their interfacing capabilities to a variety of applications.

Readers and reviewers alike have praised Andy Staugaard's book for its clarity and thoroughness. The aspiring experimenter needs only a working knowledge of Color BASIC programming and the binary number system (reviewed in the Appendix) to embark on a delightful journey toward proficiency in interfacing. The reader is shown how to construct input/output (I/O) ports and to use them to connect the computer to the mostly analog world that lies outside.

Model CoCo-100, Interface Breadboard Package, is priced at $51.25, a 10% reduction from the cost of the individual components, plus $2.50 shipping. Virginia residents add 4% sales tax. VISA and Master Cards accepted. For purchase or further information, contact Group Technology, Ltd., PO Box 87, Check,VA, 24072, Tel. 703-651-3153. ■