Use of a Computer to Design Character Recognition Logic

R. J. EVEY†

The System

THE IBM 1210 Sorter/Reader recognizes characters printed in a specified location on paper with magnetic ink.[1] A schematic diagram of the machine system is given in Fig. 1. The characters first come to a writing head which induces a magnetic field in the special purpose ink with which the characters are written. Next this magnetic field is sensed by a multi-channel reading head. The utput of the reading head is a set of ten time-dependent voltage waves.

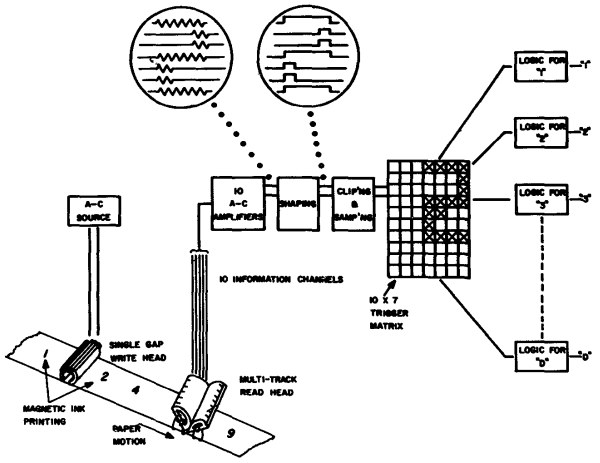


Fig. 1—System schematic.

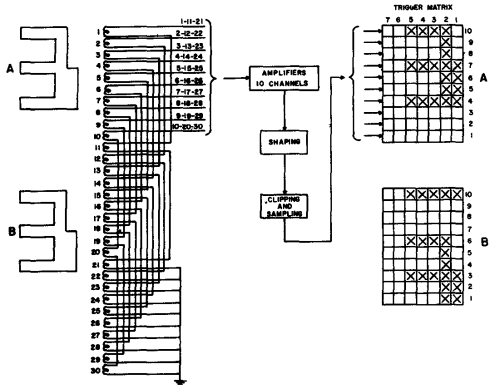


Fig. 2—Roll problem.

Actually (as Fig. 2 shows) there are thirty channels in the reading head. However, every tenth channel is "OR'ed" together (*e.g.*, 1-11-21, 2-12-22, etc.) so there are only ten outputs. These waveforms are time-sampled and changed into binary pulses by the quantizing circuits. The output of each quantizer is seven bits of binary information per character. The outputs of the ten quantizers (one per output channel of the reading head) are stored in a 10×7 trigger matrix.

The final section of the system is a set of 14 logical circuits (one for each character of the ABA alphabet[2]) made up of standard digital computer AND and OR components. These circuits are driven directly by the trigger matrix and operate in parallel. If a pattern in the trigger matrix satisfies any one of the logical circuits (called logics in the sequel), the corresponding character trigger is set. Recognition occurs if one and only one of these character triggers is set; otherwise the pattern is rejected.

It was mentioned previously that the thirty channels in the reading head are OR'ed together in groups of three. This means that the registration of the pattern in the matrix is unknown. So the system looks for recognition ten times per pattern; that is, it tries to recognize the pattern in the position in which it first appears in the matrix. Then the whole pattern is moved up one row at a time, with any bits in the top row being brought down into the bottom row. Thus each pattern really presents ten different patterns to the logics. Only after a pattern has "rolled" through all ten positions are the fourteen character-triggers examined for recognition or rejection.

This paper deals only with the design of the fourteen logics in this final part of the machine. It will attempt to make clear the problems which we tried to solve in this design and the methods we used to develop these circuits.

The Problem

The total number of different patterns possible in a 70-bit matrix is 2^{70}; and the total number of logics that can be designed for this input is $2^{2^{70}}$. The size of these numbers requires that some simplification be found to make the logical design tractable. Much of the required simplification lies in the two-dimensional

† International Business Machines Corporation, Poughkeepsie, New York.

[1] K. R. Eldredge, F. J. Kamphoefner, P. H. Wendt, "Automatic Input for Business Data Processing Systems", *Proceedings of Eastern Joint Computer Conference* (December 10–12, 1956), p. 69.

[2] Bank Management Commission: American Bankers Association, "The Common Machine Language for Mechanized Check Hand.ing", *Bank Management Publication 147, Automation of Bank Operating Procedure*, 12 East 36 Street, N. Y. (April, 1959).

correlation of bits in the matrix; that is, most of the logically possible patterns do not look anything like a possible character pattern. We found that a basic set of about 20 to 30 different patterns are obtained 90% of the time a given character is scanned. Almost all of the rest of the time a pattern is obtained which differs in one, two, or three bit positions from one of the patterns in the basic set. If these noisy bit positions are treated as don't-care positions, logical combinations of the common logical characteristics of the patterns in the basic set can be formed which will recognize virtually all the patterns obtained from scanning a character. Noisy bit positions for a given character account for over half the matrix, but this is not serious because four bits actually overdetermine the entire set of 14 characters.

The problem is thus reduced to that of finding the stable combinations of bits for a given character. At this point, however, we must consider the problem of registration — a problem which is present in all character recognition systems. Some are designed from the point of view that this is the major problem of character sensing and must be eliminated entirely; that is, an attempt is made to design the system so that once the first character is found there is no further problem occasioned by registration. In the 1210 system, however, even after the character has been scanned by the reading heads, the registration of the pattern in the matrix is unknown. A solution to the horizontal problem is the requirement that the leading edge of the character be located in the right-hand column of the matrix. The E13B font, with its strong leading edges, is designed for this.

The problem of vertical registration reduces to the "roll" problem, and the main problem here is that of cross-recognition. A degraded two, for example, may "roll" around to make a pretty good five (it should be noted that in the 1210 system this situation would result in a reject rather than a substitution, because both the "two" and "five" character triggers would be set). Part of the solution to this problem lies in the fact that the normal pattern is only eight rows high. Therefore, a condition which required at least one blank row at the top or at the bottom of the matrix was made a part of each logic. Once the pattern has been restricted so that it can move only a few rows vertically in the matrix and cannot roll completely around, the problem of design of the logics has been reduced to the required degree.

Solution

Theory

We assumed that the set of patterns to be recognized could be *approximated* by the union of two other sets of patterns which we could construct. The first of these would be the set of all admissible patterns assuming ideal printing and machine operation; that is, if the edges of characters were not ragged, there were no voids and no splatter, the magnetic field induced was uniform over the whole character, etc.

This set was generated by a program which we called the Theoretical Shape Program (TSP). Details of its operation can be found in Appendix 1. Letus say briefly here that the input to the program was a coding of each ABA character into binary bits. Each bit represented one square mil of ink. Hence, E13B characters, which are nominally 117 mils high and 91 mills wide, were entered into the 704 in the form of about 500 36-bit binary words (allowing for some blank border). This "micro-matrix" was then "scanned" by a program which simulated the operation of the reading head and quantizers. The output was a set of 10×7 "macromatrices" (*i.e.*, simply a set of patterns for each character) which were written directly onto 704 tape. The program assumed that registration, variations of magnetic density from character to character, timing across the character, fringing of the magnetic field, printing tolerances, etc. (see Appendix 1 for complete list of parameters), cannot be held firm. Hence, these "theoretical variables" were varied in the program and used to generate a set of different patterns for each character. This set was called the theoretical shapes.

We resorted to experiment to get a feel for the less systematic problems (such as voids). A hardware model of the scanning and quantizing part of the system was constructed and tied into an IBM 519 Reproducing Punch. This "print tester" scanned single characters from checks run at 1210 S/R speed and punched the resulting pattern into an IBM card. A small sample (about 10,000 checks per character) of printing chosen to cover the range of ABA printing specifications[3] was scanned and punched into one card per pattern. The resulting patterns (called "real life" shapes) were transferred from cards to tape and used to indicate the types of "noise" which might be expected to degrade the theoretical shapes.

We now had two sets of patterns (each stored on its own IBM 704 tape). Each of these was now reduced to a set which was composed of only the unique shapes of the original. These patterns were now examined by a second 704 program called the Logic Processing Program (LPP — see Appendix 2 for complete details). This program accepted, as input, logics (*i.e.*, logic statements) punched into cards in a "Boolean" notation. It interpreted each logic and stored it in core memory; then one pattern at a time was read from tape and tested against the logic. If a pattern which represented a two, say, were being tested against a logic which was supposed to recognize two's (self-test), and if the pattern was not

[3] Bank Management Commission, op. cit.

recognized by the "two" logic, but met a preset number of conditions (see Appendix 2), the pattern was printed. If it met the logic, that fact was simply noted in summary tables printed at the end of a run. If a two were being tested against a logic which was supposed to recognize, say, fives (cross-test), the criteria for printing the pattern or entering the tables were nearly the reverse of those for self-testing.

Method of Designing Logics

With these tools at hand, the following method was used to design the logics. A simple trial logic consisting of single black (1) or white (0) bits was tried against the set of theoretical shapes for that character (*i.e.*, a self-test was run against theoretical shapes). After several trials it is possible to determine a set of 10 to 15 positions consisting of single black bits inside the character outline and single white bits close to the character outline. It must be emphasized that it is always a *set* of "sure bits" which is found. For different criteria a different set will be found. For example, a program was written which determined the maximal set of sure bits for each theoretical character. However, in some cases, a more desirable set of sure bits would be one which distinguished sharply a given character from that character (or characters) which looked the most like it. These "sure bits" were then used as a trial logic for running a cross-test against the rest of the theoretical shapes. The result of this run would be a reduced set of "sure-bits", which were useful in telling this character apart from the other theoretical characters. Then these "useful sure-bits" were used as conditions for a trial logic for the given character.

First, this trial logic would be self-tested against corresponding real-life shapes. Samples of real-life shapes would not be recognized because of voids, ink-splatter, skew, etc. By examining the tabulations and patterns printed by LPP, the designer would attempt to modify the single-bit trial logic by OR'ing a more complex condition to the sure-bits which gave trouble. This new logic would again be real-life self-tested. After a number of trials, a logic would be obtained which would recognize all of the real-life shapes the designer felt were realistic. Then the logic was real-life cross-tested and modified using a similar procedure. Here, however, the criterion for final acceptance was that *no* character should be misrecognized by the logic (this was due, of course, to the system's more stringent requirements on substitutions than rejections). A flow-diagram of the above procedure is shown in Fig. 3.

Several modifications of each logic would usually have to be made at each step in the process before the logic would be considered satisfactory. Sometimes it was necessary to start from the very beginning with a search for a new set of useful sure-bits. In all cases a complete, transmissible record of the design of each trial logic, the results obtained in testing it against the trial shapes, and the reasons for modification existed in the summaries kept by LPP.

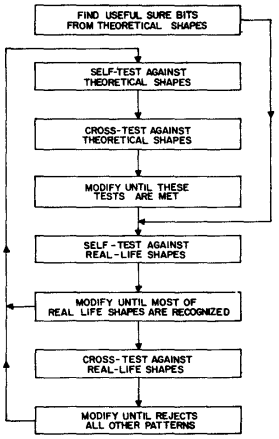


Fig. 3—Statement writing procedure.

Conclusion

Only two other methods of designing logics of this type are known to the author. One of these consists of building hardware which allows the engineer to shift wires in the model quickly (somewhat like IBM plugboards for EAM equipment). In this way logics can be wired directly into the machine and paper can be fed through an actual model of the system. This method has the advantage that the engineer knows the logics are trying to recognize patterns which are produced under field conditions. It has the great disadvantage that there are no records of patterns successfully recognized by the logic. When a change is made in logic wiring and a retest is run, the engineer has no way of knowing whether the same patterns as before are being presented to the logic. Hence, he has no assurance that he is really comparing the new logic against the old. The new logic may work better; but it may be because it is seeing more easily recognizable patterns. This method of designing logics has been tried at IBM and has not been as successful as the subject method in either time, cost of logics, or reliability. However, using the procedure described in this paper, a set of statements for each of the fourteen characters was developed with an expenditure of six man-months for the 704 programs (which are of an exceedingly general nature and have been used in whole or part for other applications) and two man-months for the design of the logics. Further, it was found that two different designers working independently on the same statement tended to produce logics that were equivalent in cost, performance, and the bit positions used (see Fig. 4). The best proof of the method, however, lies in the fact that the initial set of statements developed through its use have been wired into models of the

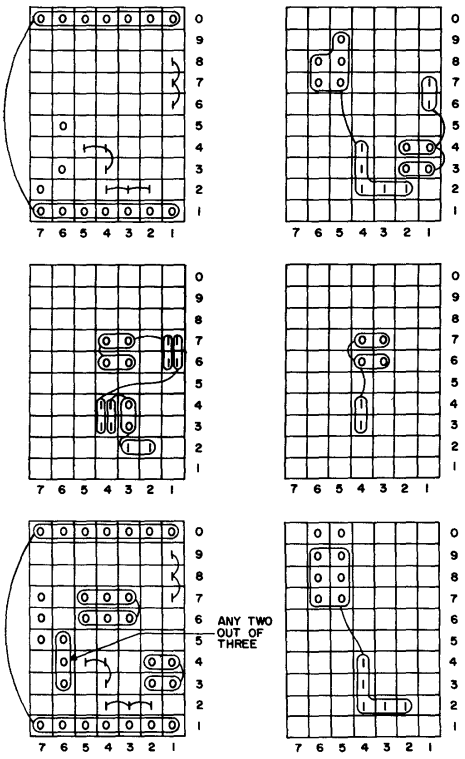


Fig. 4—Two "2" logics.

1210 S/R and have remained there unchanged after more than a year of rigorous testing.

Another method known to the author is that of devising an automatic procedure to design these logics. Most exhaustive procedures can be ruled out due to the astronomical number of possible logics, but useful procedures have been developed by limiting the complexity of the conditions used in the statements.[4] However, possibly because of this limitation, statements so produced have never been as successful in practice as those designed by people.

There was a time, nevertheless, when we felt that a definite short-coming of this method was that it was not automatic. In the many areas in which there is an attempt being made to utilize computers for the solution of complex decision problems (*e.g.*, theorem-proving, language translation, network analysis and synthesis, etc.), the goal is complete automation. However, this was not our goal. We needed a reliable set of logics and we were able to utilize the computer to advantage in completing this task. It processed a trial logic against a controlled set of input patterns. It ran tests and tabulated the results of this processing. Under a variety of sense-switch controls it displayed specific items of interest to the designer. Finally, it kept accurate records of this continuing iterative process of logic design, so that previous work could be re-examined. In this way the human beings in the process were freed from monotonous tasks and could devote their experience and creative judgment to the actual task of designing logics that recognize characters.

[4] P. H. Howard, "A Computer Method of Generating Recognition Logics for Printed Characters," *IBM Technical Note*, TN 00.10070. 357 (May 5, 1959)

Acknowledgments

It would be impossible to mention all the people who have helped with this project. But at the risk of omitting equally worthy contributors, I would like to single out D. R. Andrews, G. E. Bartholomew, P. C. Murley, and L. O. Nippe, who wrote most of the programs; A. J. Atrubin and P. H. Howard, whose constructive criticism and helpful suggestions contributed both to the writing of the programs and to the design of the logics; E. C. Greanias and A. Hamburgen, who made significant contributions in the earlier stages of this project; and especially Dr. J. P. Lazarus, without whom neither the project nor this paper could have been completed.

Appendix 1 — Details of TSP

Input to the program consisted of two sets of IBM cards. The first set consisted of a coding of the character into one-mil squares. This was accomplished in this fashion:

A detail drawing of the character was blown up to 50 times life-size. A grid marked off in squares, which represented one square-mil to the same scale, was then laid over the character. Each coordinate on this grid was marked. Hence, a person could quickly see the coordinate where each row started into black and where it left. One card was then punched per row — with first the coordinate when black was started, when it was left, when it started again (if it did) and so on. Since each ABA character is 117 mils high nominally, this would result in 117 cards per character. A further coding was incorporated, however, in that where the edges of the character are not curving, one card may be the same as a preceding card. Hence, there is no need to repeat the next card; simply punch into the first card the number of times it is to be repeated. Fig. 5 gives the listing of the cards required to code the character 2. These cards were read by the program (actually they were put on tape and read from there) and interpreted into bits where there was black indicated in the character and blanks where the character was white.

The second set of cards (an example may be seen in Fig. 6) contained a complete set of the parameters which could be varied in the program. These parameters (and the card fields in which they were punched) were:

(1) The dimensions of the macromatrix (*i.e.*, the output matrix or trigger matrix of the S/R).

	IDENTIFICATION	ROW	NO. OF REPEATS	BLACK STARTS	BLACK STOPS	BLACK STARTS	ETC.
CARD COLS.	1,2	3-5	6-8	9-11	12-14	15-17	ETC.
	D2	117	001	005	048		
	D2	116	001	003	050		
	D2	114	002	002	051		
	D2	109	005	001	052		
	D2	107	002	001	051		
	D2	106	001	001	050		
	D2	105	001	001	048		
	D2	104	001	001	017		
	D2	103	001	001	015		
	D2	101	002	001	014		
	D2	070	031	001	013		
	D2	068	002	001	014		
	D2	067	001	001	015		
	D2	066	001	001	017		
	D2	065	001	001	048		
	D2	064	001	001	050		
	D2	062	002	001	051		
	D2	057	005	001	052		
	D2	055	002	002	052		
	D2	054	001	003	052		
	D2	053	001	005	052		
	D2	052	001	036	052		
	D2	051	001	038	052		
	D2	049	002	039	052		
	D2	018	031	040	052		
	D2	016	002	039	052		
	D2	015	001	038	052		
	D2	014	001	036	052		
	D2	013	001	005	052		
	D2	012	001	003	052		
	D2	010	002	002	052		
	D2	005	005	001	052		
	D2	003	002	002	051		
	D2	002	001	003	050		
	D2	001	001	005	048		
	END OF DATA FOR GIVEN CHARACTER.						

Fig. 5—Coding for Character 2 for TSP.

(2) The font (this would be varied by changing the first set of input cards).

(3) The width of a reading channel to the nearest mil.

(4) The width of the dead space between the channels to the nearest mil.

(5) The horizontal sampling interval in mils.

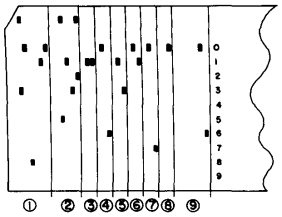
(6) The clipping level of the quantizing circuits (*i.e.*, the height of the voltage waveform they would have to see to call it above the noise level.)

(7) The integration time of the quantizing circuits.

(8) The initial registration of the character (that is, whether its leading edge were sensed too soon due to magnetic fringing or other effects, right on time, or late due to missing or low-density ink).

(9) Printing tolerance.

The program would first read a set of character coding input cards, interpret them, and position the coded character in storage in such a way that it simulated a character with its bottom edge on the bottom edge of a reading channel. Then a parameter card would be read and the character "scanned" in accordance with the parameters punched therein. The result of this "scan" would be a pattern (or macromatrix) which was written on tape immediately. Then the character would be "moved" (or "rolled")



FIELD	COLS
①	1-6
②	7-12
③	14-15
④	17-18
⑤	20-21
⑥	23-24
⑦	26-27
⑧	29-30
⑨	35-37

Fig. 6—Parameter card for TSP.

up one mil in its relation to the channel and land (dead space) and again scanned in accordance with the same set of parameters. This process would continue until the character had rolled up to the position in which its bottom edge just rested on the bottom edge of the next higher channel. At this point it is obvious that we would begin to see the same set of patterns all over again. So another parameter card would be read and this process repeated for that card. This would continue until all the parameter cards for a given character were read, at which point a new set of character coding cards for the next character would be read and the whole process repeated. This process is illustrated in the simplified flow-chart of the program shown in Fig. 7.

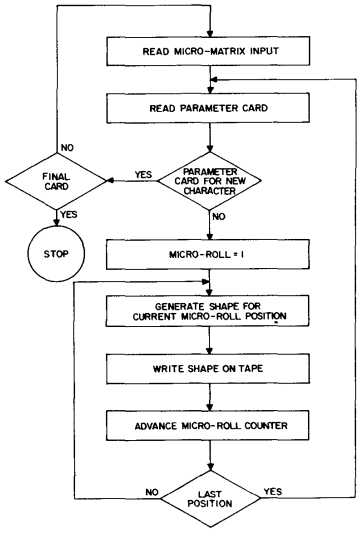


Fig. 7—TSP flow chart.

There was one parameter which does not appear in a parameter card. That is the system of quantizing used. This was varied by reprogramming. That part of the program was made into a closed subroutine and reprogrammed whenever the engineers changed their quantizing circuits. About five different types of quantizers were tried and they had so little in

common we felt this was better than attempting a general program. It should be mentioned here that after the program was used a couple of times, it was so successful in simulating the scanning that the engineers would try a new idea for quantizing here before they would try it in hardware.

Appendix 2 — Details of LPP

The input to LPP consisted of two parts also. First, of course, was a set of cards into which was punched the logic to be tested. These were punched in this manner:

The character for which the logic was written was punched in column 1, (A, B, C, D being used for the four special symbols of the ABA alphabet). The number of conditions was punched in columns 2 and 3. A condition is a multistaged logical AND'ing and OR'ing of trigger matrix bits which, when AND'ed with other conditions, forms the logic for the given character. No assumption of minimal form is made, so that the same logic may be decomposed in different ways into conditions. For example, if A and B are two conditions, the total logic consists of $A \cdot B$ and 02 is the number of conditions. AB may be taken as a condition and the total logic then has one condition. Suppose $A = C + D$, then there are two conditions, $(C + D) \cdot B$; or the logic can be written $BC + BD$, which is only one condition. Hence reference is most easily made to a logic picture to see what was constituted as a condition. Fig. 8, which shows a simple logic and what would be punched into the logic card, may make this clear.

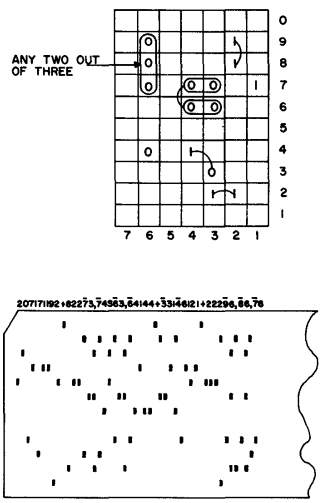


Fig. 8—Simple logic.

Starting in column 4, a cycle of symbol-row-column started and kept up until column 72 or until all the logic was punched. If the logic had to extend over to a second card the same sequence was used; that is, character, total number of conditions, symbol, row, column, etc., starting where one left off on the preceding card. The symbols used were numerals 1 to 9, "+" for OR, a "," comma for AND, and the letter "S", which also symbolized a logical OR but had a larger scope than the plus sign. The numerals indicated that a new condition was starting and told how many of the subconditions following it were to be satisfied (2 out of three for example). A subcondition is one bit specified by the row and column location. If the bit were to be a blank, a negative sign (X over-punch) was punched over the row.

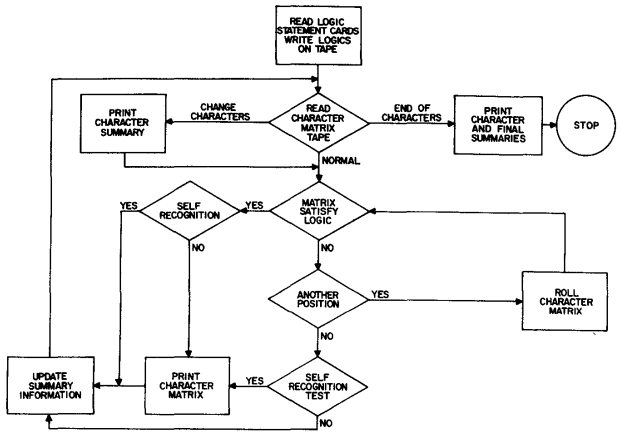


Fig. 9—Logic processing program.

```
FREQ. TABLE   2 LOGIC   CHAR. 2        490 NCR     MFC  486      CNML 2

                                                    A
                    10   54   55   55   44         9
                    12   86   93   96   99         8
                     0    9   10   39  100         7
                    10   56   58   64  100         6
                    13   92   98   97   97         5
                    13   98    8    4    4         4
                    13   93   69   60   48         3
                    12   91  100  100  100         2
                                                    1
     8    7    6    5    4    3    2    1
POS / FREQ.   1 /  486,
```

Fig. 10—Frequency table (FT).

These cards were read by the program, interpreted, and stored in memory. (See Fig. 9 for a flow-chart of LPP.) Then the program read one character pattern (the second element of input) from tape. This pattern was tested against the logic. As we have said, if the character were being tested against its own logic and met all the conditions this was noted in a final summary table. Actually more was done with it. The whole pattern was added, a bit at a time, into a *frequency table* (Fig. 10). That is, this table kept track of how many times the characters had bits in each matrix position when considered in the roll position in which they were recognized by the logic. Now, if the pattern was not recognized, it was rolled through all ten roll positions, and the program kept track of the roll position in which it missed the fewest number of conditions (or the first position in case of a tie).

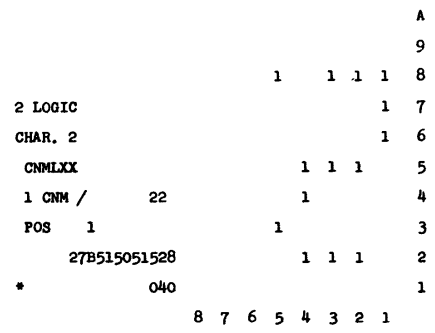


Fig. 11—Printed pattern (PRAT).

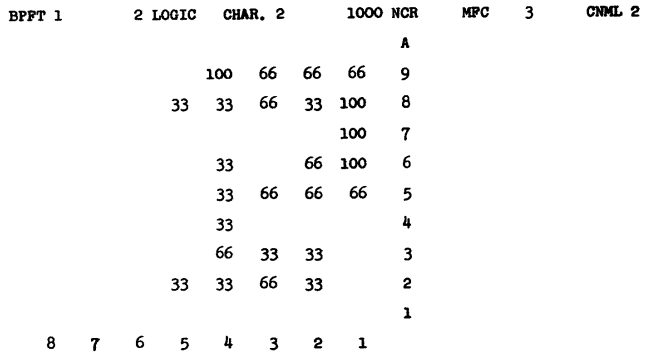


Fig. 12—Best position frequency table (BPFT).

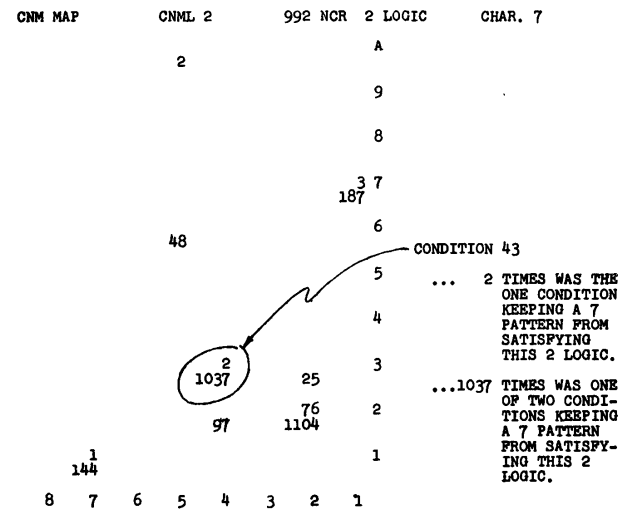


Fig. 13—Condition-not-met map (CNMM).

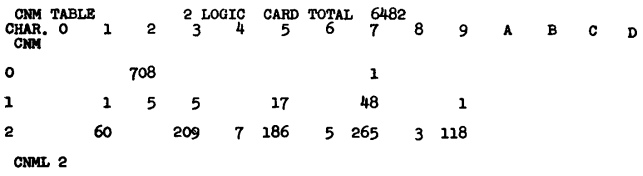


Fig. 14—Summary.

Then the whole pattern would be printed out (these printouts were called *printed patterns*, or PRAT's — Fig. 11). Further, the pattern was added into a table called a *best position frequency table* (Fig. 12). Further, a table was made up of the conditions which were missed. These were called *condition-not-met-maps* (Fig. 13) and told the conditions which kept patterns from recognizing. If the pattern came within one condition of recognition, the count was printed on one line; but if it were two or more conditions from recognition, the count was printed on a different line.

As has been said previously, if a pattern were being tested against a logic for a different character all the above tables were entered but the criteria for entrance were simply reversed. Further, entrance was made in the tables for every roll position. In this way the CNMM (condition-not-met-maps) told what conditions were actually keeping characters from being recognized. All of these tables were printed at the end of each character run. Only a final *summary* table was printed at the end of the complete run (Fig. 14). This told for each character how many patterns came within 0 (*i.e.*, complete), 1, or 2 conditions of being met.

Complete control of entrance into each of the summary tables and printing of the summaries was maintained by using a combination of control cards and sense switches. The control cards specified whether or not a certain summary was to be kept and, if so, gave a limit of conditions. If a pattern missed recognition by more than this number of conditions, the summary table for that character was not entered. Then, as the program ran, the logic designer could choose to see certain tables (or even change the course of the program) by a selection of sense-switch settings. In this way the program displayed only that data the designer thought would be helpful at any given time.