SYMBOLIC PROGRAMMING

N. Rochester

August 3, 1952

Abstract

Automatic calculators can be programmed to interpret programs which have been written with symbolic instead of actual addresses. This method allows the calculator to assume much clerical burden which must otherwise be borne by the programmer.

SYMBOLIC PROGRAMMING

Programs for automatic calculators can be written with symbolic addresses instead of actual addresses. The automatic calculator can then, by means of a method to be described in this paper, read a symbolic program and convert it to an ordinary program with actual addresses. Three important advantages result from the use of this procedure.

a) The calculator assumes part of the clerical burden which must otherwise be born by the programmer during the initial writing of a program.

b) The programmer can insert modifications, additions, or deletions with very little effort and little chance of error. This can be done after the program has been tested or even used on the calculator.

c) The calculator can assume almost all of the clerical burden of incorporating library programs into newly written special programs.

While this method is applicable to many automatic calculators, it will be described here in the form that it takes when used with the IBM Type 701 and its associated units. This is a large-scale, binary calculating installation which uses single-address instructions. A more complete description [1] of this calculator is given in the proceedings of the Pittsburgh Meeting of the Association for Computing Machinery.

The programmer's problem is illustrated by the program shown in Fig. 1. This is a piece of a program written with actual addresses. After writing the original program, the programmer decided to insert an instruction between 0289 and 0290. This insertion has changed some of the succeeding addresses. It has not only changed the location of the following instructions, but it has also modified some of them.

In a long program, changes of the sort shown in Fig. 1 are impractical. It is altogether too much work to go through several hundred instructions, modifying every location and discovering every single other change which is necessary. When confronted by this problem, most programmers use a well-known devious expedient. This expedient avoids the renumbering problem but wastes memory space and leaves a tangle of logic which makes the program more difficult to read.

Consider, now, the symbolic program shown in Fig. 2. This is the same program as the one shown in Fig. 1, but it has been written with symbolic addresses. Some of the addresses in Fig. 2 are four-digit numbers while others are sequences of numbers punctuated with periods. The four-digit numbers are actual addresses while the punctuated numbers are symbolic addresses. The symbolic addresses are really symbols which stand for unspecified actual addresses.

The numbers in a symbolic address have no numerical significance. A symbolic address is merely a name. The periods which punctuate a symbolic address are not decimal points, but are merely markers. They serve as a guide to the programmer as he works but are not used by the calculator.

In Fig 2, the insertion did not have repercussions throughout the program because the symbolic addresses have no numerical significance.

One way to use these symbolic addresses is to use the number before the first period to indicate major breaks in the logic of the program. In other words, the programmer breaks his program into blocks of instructions and gives each block a different number. The number after the first period can be used to number the instructions within a single block. The number after the second point can be used for insertions such as the one shown in Fig. 2.

It will be shown that the calculator can accept programs written in this form. The calculator will not care about the conventions which the programmer has used provided, only, that the programmer has managed to avoid using the same symbolic address for two different purposes.

The system which has been described above is merely a minor modification of the system proposed by Burks, von Neumann and Goldstine, [2] in 1947. The principal contribution of this paper is to show how the calculator can be made to do the work of assigning the actual addresses. In the earlier system, this work was left to the programmer. Some other work along this line has been reported, [3,4] but these other systems do not seem to allow the easy use of library programs nor do they provide the programmer with a detailed actual program so that he can know exactly what the calculator thinks it is supposed to do.

This system has one further class of symbols which are allowed. These are the 26 characters of the alphabet with subscripts and superscripts. This last convention is particularly convenient for representing the addresses used to store the elements of a matrix or entries in a table.

In order to use this system the programmer writes his program as shown in Fig. 2. Then the instructions are key punched on IBM cards in symbolic form. One instruction is punched on each card. Fig. 3 shows the symbolism which is used in punching.

The cards are then arranged in a deck in the order in which the programmer wants the instructions to appear in memory. This deck, to-

gether with the program deck is given to the calculator which produces a printed copy of the program with symbolic and actual addresses side-by-side. Simultaneously the calculator punches a deck of cards containing the finished program ready to go into the calculator.

Fig. 4 shows a real program which was written in symbolic form. Fig. 5 shows the printed program which the calculator produces for people to read, and Fig. 6 shows the binary card on which the calculator records the program for its own subsequent use.

Notice, in Fig. 5, that the original program is written with symbolic and decimal addresses while the calculator prints the actual program with octonary addresses. This is desireable since people are trained to think in decimal, but when they are observing the action of a binary calculator they need a representation, such as octonary, which is easily translatable to binary.

Notice, also, that the calculator prints comments along with the numbers. These are the comments which the programmer writes in his original program sheet in order to make his program more readable. They are key-punched on the original cards and read and saved by the calculator. Because of these, the form which the calculator prints is all ready for duplication and circulation to other people.

In order to describe how the assembly program works, it is convenient to proceed with easy steps and consider successively more complicated aspects of the problem, rather than jump headfirst into all of the details.

SHORT PROGRAMS

After a program has been written, using symbolic addresses, a deck of cards is punched with one instruction or number per card. The programmer must then arrange the cards in the same order in which he wants the words in memory.

The assembly program is then put in the memory and the newly prepared program deck is run in. As the calculator reads each card, it assigns the actual address at which the instruction or constant is to be eventually located. It then stores, in the memory, the symbolic location, the newly assigned actual location, the word itself, and some identification.

When all of the instructions have been stored, each symbolic address which represents a location of an instruction or constant is associated with a corresponding actual address. However, the address parts of some instructions are symbolic and these have not yet received actual addresses.

The calculator next looks at each symbolic address. Whenever it finds a symbolic address with no actual address, it interrupts the sequence and makes a search to find this same symbolic address somewhere else with an actual address. It then attaches this actual address to the symbolic address which needed it and goes on.

The calculator then makes a third pass through the information and prints one word per line printing all numbers and addresses. At the same time the instructions are compressed into a single unit record, discarding all of the extra information which is needed only to calculate the actual instructions. This unit record is then punched on cards, in binary, and the process is complete.

LONG PROGRAMS

Frequently, the simple process described above will not suffice because the program will not fit in the memory with all of the excess baggage which goes along with it at various intermediate stages. This limit comes at 384 instructions if the calculator is equipped with the full sized, 2048 word memory.

It is necessary to consider means for breaking the program into pieces which can be treated separately. The primary difficulty in doing this is that the process described in the above will not always work because an instruction may have, as its address part, the address of a word which is not in the memory at the time. The calculator will have to use part of the memory as a file of undetermined addresses where it can make a note of the addresses which it needs so that it can find them when it gets a chance.

The procedure is that the programmer puts the program cards in order as before. He then inserts heading cards to divide the deck into reasonably sized batches. These breaks should be inserted where there will be a minimum of cross reference.

The calculator then reads the cards as before but stops when it reaches the first heading card. It then goes through its file of instructions and addresses, and whenever it finds a symbolic address without an actual address it makes a search to see whether this symbolic address appears anywhere else in memory with an actual address. If it fails to find an actual address it adds this symbolic address to the undetermined address file, (unless, of course, the undetermined address file already contains that particular symbolic address). Having accomplished as much as it can with this batch of instructions, it records this batch on tape and goes on to the next batch.

Having completed all the batches, the calculator rewinds the tape and starts over again. This time it is able to get all of the addresses which it could not get before so it can print the results, one word to a line, and compress each batch into a much shorter unit record, which it punches on binary cards.

LIBRARY PROGRAMS

Frequently, one wants to incorporate library programs into larger programs or to combine two independently prepared programs. The symbolic address system which has been described leads directly to a method for handling this problem.

This practice has not proved to be especially fruitful in the past but it should be more useful with this symbolic address system because:

a) The Programmer can conveniently modify, delete, or insert instructions in the library programs before giving them to the calculator to use.

b) The calculator can assume the main burden of adapting the programs to each other.

Suppose that the programmer wishes to assemble two or more independently written programs.

If no precautions were taken, there would usually be homonyms. "1.4.2" might mean one thing in one program and something else in another. The homonyms are eliminated by attaching a two-digit prefix to each symbolic address. A different prefix is used for each independent program. These prefixes are introduced on the heading cards which also serve to divide the instructions into batches as described in the above.

Synonyms will occur at the points of contact between independently written programs. The programmer must choose which of the synonyms are to be kept and which are to be discarded and must inform the machine of his choices. It is by doing this that the programmer states the relation between the programs. For example, if a programmer has a sin x program and a log x program, then by different choices of synonyms he can get a combined program for log (sin x) or sin (log x).

The synonyms are entered with one pair to a card. On the first pass, when it is reading the original cards, the calculator discards unwanted synonyms.

Except for these two minor complications, homonyms and synonyms, no difficulties are introduced when one uses library programs.

The IBM type 701 can do this kind of program assembly at a rate of nearly 75 instructions per minute. It reads the original cards at the rate of 150 per minute and it prints the final report at the rate of 150 lines per minute. Most of the running of tape and calculating is done in the spare time during card reading cycles and printing cycles. Only occasionally does the calculator have to interrupt the succession of card or print cycles for other purposes, and then only for a few seconds.

The assembly program which is now in use is not fully self-checked. A revision which is now under way will be completely checked by arithmetical processes. A description of the checking has not been included in this paper because the additional complexity would have interfered with exposition and because it is not often convenient to transplant checking schemes from one calculator to another of different design.

This system of writing and using programs with symbolic addresses is a powerful tool for the experienced writer of complicated programs. It is not a very good system for the novice because it requires knowledge which is hard to acquire without some experience with a simpler system. It is not a good system for very short simple programs since it requires too much apparatus. These simpler problems are easily enough handled by writing directly in the machine language.

While this system offers some advantage in the original writing of a program, it becomes of greatest importance when programs have to be modified or when the programmer wants to steal parts of an old program in order to write a new one.

This system was conceived and worked out for a group whose programs are usually either utility programs, diagnostic programs, or programs which enable the IBM type 701 to simulate other devices. In each of these cases the programmer is continually faced with the necessity for inserting small or large changes in programs which are already in use. The technique may be useful to mathematicians if they also have frequent need to modify programs.

Another IBM group, also using the IBM 701, has worked out a different assembly technique [5], which requires less of the calculator and is conceptually simpler. It is preferable for people who do not often have to modify existing programs.

Nathaniel Rochester

FOOTNOTES

1
M. M. Astrahan and N. Rochester, "The Logical Organization of the New IBM Scientific Calculator", Proceedings of the Pittsburgh Meeting of the A. C. M., May 2-3, 1952.

2
H. H. Goldstine and J. von Neumann, "Planning and Coding of Problems for an Electronic Computer, Part II, Vol. I" Institute for Numerical Analysis, Princeton, 1947.

3
J. W. Carr, "Discussion on the Use and Construction of Subprograms", Proceedings of the Pittsburgh Meeting of the A. C. M., May 2-3, 1952.

4
M. V. Wilkes, "Free Address System of Coding", Review of Electronic Digital Computers, Joint AIEEIRE Conference, p. 114; February 1952.

5
E. F. Codd, W. F. McClelland and P. W. Knaplund, "Regional Programming", Proceedings of the Toronto Meeting of the A. C. M., September 8-9, 1952) not given

Location of Instruction	Instruction: Operation Part	Instruction: Address Part
----	-----	-----
0288	STORE	0868
0289	R ADD	0702
0291 ~~0290~~	ADD	0750
0292 ~~0291~~	STORE A	0293 ~~0292~~
0293 ~~0292~~	R ADD	(0000)
----	----	----
0290	A LEFT	0001

Fig. 1

An Actual Program Which Has Been Modified

Location of Instruction	Instruction: Operation Part	Instruction: Address Part
----	----	----
11.8	STORE	14.9
11.9	R ADD	13.4
11.10	ADD	15.8
11.11	STORE A	11.12
11.12	R ADD	(0000)
----	----	----
11.9.1	A LEFT	0001

Fig. 2

A Symbolic Program Which Has Been Modified

Conventional Symbol	Symbol Written on Program Paper	Symbol Punched on Cards
1.1	1.1	010100
1.2	1.2	010200
1.10	1.10	011000
1.10.3	1.10.3	011003
1.11	1.11	011100
C	C	0C0000
C_1	C 1	0C0100
C_{11}^{23}	C11.23	0C1123
${}^{2}C_{11}^{23}$	2C11.23	2C1123

Fig. 3

Symbolism on Paper and on Cards

Read - Write Subprogram

Location of Instruction	Operation Part	Address Part	Comment
2.1	Add	4.3	
2.2	Store Address	2.8	Store location of first address
2.3	Add	4.4	
2.4	Store Address	2.10	Store location of number of words
2.5	Store Address	3.10	Store location of end of file procedure
2.6	Add	4.4	
2.7	Store Address	3.12	Store loc. of end of record procedure
2.8	Reset and Add	0000	
2.9	Store Address	3.8	Insert first address for copy
2.10	Reset and Add	0000	
2.11	Shift Left	0001	
2.12	Store	4.1	Store number of words
2.13	Transfer on Plus	2.16	
2.14	Reset and Subtract	4.3	
2.15	Transfer	2.17	
2.16	Reset and Add	4.3	
2.17	Store	4.2	Store address increment
2.18	Transfer	3.8	
3.1	Reset and Subtract	4.2	
3.2	Add	3.8	
3.3	Store	3.8	Store next address with copy
3.4	Reset and Subtract	4.2	
3.5	Add	4.1	
3.6	Store	4.1	
3.7	Transfer on Zero	3.11	Is the record complete
3.8 -	Copy	0000	
3.9	Transfer	3.1	
3.10	Transfer	0000	End of file procedure
3.11	Write	2052	Delay until end of record
3.12	Transfer	0000	End of record procedure
4.1	Stop	0000	Index, 2N or TR OV 2N
4.2	Stop	0000	Space for maniuplation
4.3	Stop	0002	
4.4	Stop	0001	

Fig. 4

An Original Program

```
04.01.00 00.00.    7736 + 00 0000  INDEX, 2N OR TR OV 2N
04.02.00 00.00.    7737 + 00 0000  * SPACE FOR MANIPULATION
02.01.00 04.03.00  7740 + 11 7776
02.02.00 02.08.00  7741 + 15 7747  STORE LOCATION OF FIRST ADDRESS
02.03.00 04.04.00  7742 + 11 7777
02.04.00 02.10.00  7743 + 15 7751  STORE LOCATION OF NUMBER OF WORDS
02.05.00 03.10.00  7744 + 15 7773  STORE LOC OF END OF FILE PROCEDURE
02.06.00 04.04.00  7745 + 11 7777
02.07.00 03.12.00  7746 + 15 7775  STORE LOC OF END OF RECORD PROCEDURE
02.08.00 00.00.    7747 + 12 0000
02.09.00 03.08.00  7750 + 15 7771  INSERT FIRST ADDRESS FOR COPY
02.10.00 00.00.    7751 + 12 0000
02.11.00 00.01.    7752 + 26 0001  *
02.12.00 04.01.00  7753 + 14 7736  STORE NUMBER OF WORDS
02.13.00 02.16.00  7754 + 03 7757  *
02.14.00 04.03.00  7755 + 06 7776  *
02.15.00 02.17.00  7756 + 01 7760  *
02.16.00 04.03.00  7757 + 12 7776  *
02.17.00 04.02.00  7760 + 14 7737  * STORE ADDRESS INCREMENT
02.18.00 03.08.00  7761 + 01 7771
03.01.00 04.02.00  7762 + 06 7737
03.02.00 03.08.00  7763 + 11 7771
03.03.00 03.08.00  7764 + 14 7771  STORE NEXT ADDRESS WITH COPY
03.04.00 04.02.00  7765 + 06 7737
03.05.00 04.01.00  7766 + 11 7736
03.06.00 04.01.00  7767 + 14 7736
03.07.00 03.11.00  7770 + 04 7774  IS THE RECORD COMPLETE
03.08.00 00.00.    7771 - 37 0000
03.09.00 03.01.00  7772 + 01 7762
03.10.00 00.00.    7773 + 01 0000  END OF FILE PROCEDURE
03.11.00 20.52.    7774 + 32 4004  DELAY UNTIL END OF RECORD
03.12.00 00.00.    7775 + 01 0000  END OF RECORD PROCEDURE
04.03.00 00.02.    7776 + 00 0002
04.04.00 00.01.    7777 + 00 0001
```

Fig. 5 - Assembled Program as Printed by IBM 701

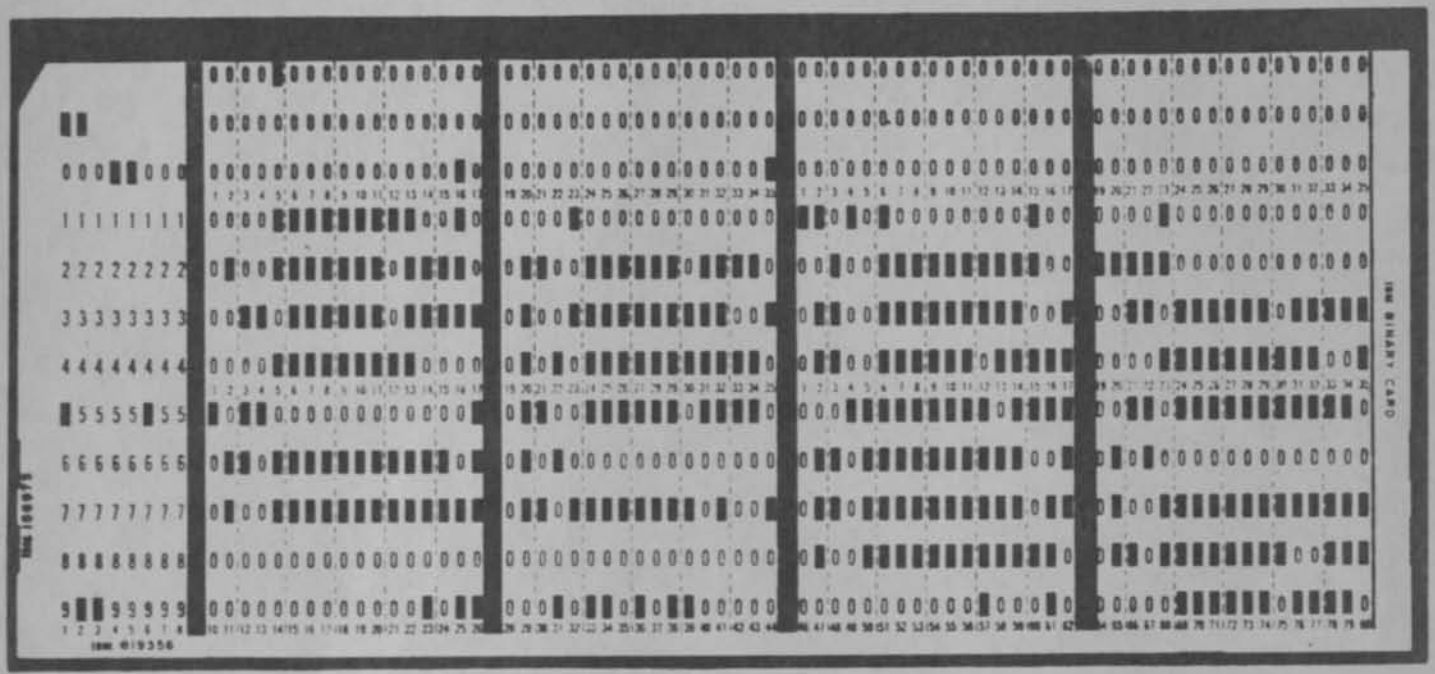

Figure 6

Program on Binary Card after Assembly