Regional Programming

In general, the successful operation of a sequence of instructions for a stored-program calculator requires that:

1. the instructions are executed at the appropriate time relative to each other and to the execution of other instruction sequences in the calculator, and

2. the instructions are in the proper storage locations at the time of their execution.

For example, consider the following sequence of instructions:

	Instruction	
Location	Op Part	Add Part
0	Reset Add	450
1	Add	500
2	Store	450
3	Reset Add	1
4	Add	100
5	Store	1
6	Transfer	0

Location 100 contains the integer 1.

This instruction sequence is introduced for illustration only. It is designed to add the contents of cells with successively higher addresses, starting with address 500, to the contents of cell 450. The instructions occupy locations zero through six, the constant "1" is available in location 100, and the sum is stored in location 450.

Three of the seven instructions in the example sequence make reference to other instructions in the same sequence. Instructions three and five are used in the modification of instruction one, and instruction six transfers control to the beginning of the sequence. Thus, instructions may refer to other instructions in order:

1. to compute or otherwise modify these instructions; and

2. to execute a conditional or unconditional transfer of control.

The effect of these self-references in a sequence of instructions is to determine the location of the sequence by its construction. It is clearly impossible, for example, to transplant the seven instructions above into new locations and have them operate correctly without modification of their address parts. This process of assigning instructions to new locations is referred to as relocation.

It is possible to write programs for a stored-program calculator using the locations from which the instructions are to be executed. This was done in the writing of the example program above, and it is usually efficient to write programs in this way when:

1. the programs are short and easy to write without error;

2. the programs require a relatively small part of storage so that the problem of pre-assigning locations to a program without knowing its extent is not troublesome; and

3. the programs will not have to be executed at a later time from other locations.

It is necessary to use actual locations in preparing programs when the calculator which will execute them is unable to relocate them. Therefore, the early operation of a calculator is usually programmed in this way, particularly since the programmers may be more interested in testing the calculator than in producing working programs for later use.

There soon comes a time, however, when the assignment, and particularly the reassignment, of actual calculator locations appearing in a program becomes so tiresome, inaccurate, and inefficient when performed by hand that a new system must be found to solve the relocation problem. This system should be designed in such a way that programs may be written to occupy arbitrarily chosen storage locations. This allows library programs to be reused without manual modification and makes it possible to have several programmers write different sections of a large program without knowing what locations are to be occupied by any of these sections. All assignments of actual storage locations are performed automatically by machine.

Consider the arithmetic which must be performed on the example sequence shown above in order to allow this sequence to produce the same result as before when relocated from locations zero through six to locations 138 through 144. The new program is shown below, with necessary changes in instruction address parts indicated to the right:

	Instruction		
Location	Op Part	Add Part	Change
138	Reset Add	450	0
139	Add	500	0
140	Store	450	0
141	Reset Add	139	138
142	Add	100	0
143	Store	139	138
144	Transfer	138	138

Location 100 contains the integer 1.

The interesting points of this example are that:

1. Some instructions must be modified, while others need not be.

2. All instructions changed differ from their previous values by the same amount.

These two results are typical of the arithmetic of relocation. In the relocation of a single type of information - instructions in the example above - by a fixed amount, only certain instructions of a given sequence are modified, but all those changed are modified by a constant increment.

In the above example of relocation, only instructions were relocated. A different result would have been obtained if it had been decided that only the location of the constant "1" should be changed. In this case instruction four is the only one requiring modification. Similarly, the initial number to be entered in the sum may be determined by assigning the appropriate address part to instruction one.

It may be seen from the above example that the relocation of information referred to by a sequence of instructions requires that each instruction referring to that information be discovered and its address part changed by the correct amount. This search and modification is routine and uninteresting to a coder, and therefore he is not apt to do the job either accurately or well. Fortunately, however, these simple operations may be performed automatically by machine.

One system that we at IBM have found to be particularly convenient in the solution of this relocation problem is called Regional Programming. This system has been used successfully for some time in the preparation of programs for the IBM Electronic Data Processing Machines, including the Type 701 Control Unit. Although the system was designed for specific application to our installation in Poughkeepsie, it is believed that the techniques of regional programming will prove applicable to other stored-program single-address machines. [1]

The basis of this system is the assignment by the programmer of an indicator for each instruction. The assignment of this indicator, called the relative index, is made at the time the program is first written. The purpose of the relative index associated with a given instruction is to provide the programmer and the calculator with an indication of what type of information the instruction is to operate upon. Therefore, this relative index is always associated with the address part of the instruction.

The examples already shown indicate the need for an indicator such as the relative index for use with instructions. The question at once arises as to whether such indicators are also required for data, results, and other pieces of information which may at some time be stored in the calculator but which are never executed as instructions. For the bulk of this non-instructional information indicators are not required, but there is one class of constants, called instructional constants, for which relative indices are necessary. These instructional constants refer to storage locations and are used for setting initial address parts into instructions belonging to iteration loops, for testing for the end of the execution of these loops, and so forth. In the example above, an instructional constant might be used to reset to 500 the address part of instruction one in preparation for the execution of the adding loop. Instructional constants must not be confused with ordinary constants, such as the integer "1" in the example above, whose value does not depend upon its location. In the discussion which follows, instructional constants will always be grouped with instructions for purposes of classification. This is convenient since the two types of information are very similar, and an instruction which may be executed in one part of a program may be used in subsequent operations to reset or test other instructions. Thus, a single word may, at different times, operate as an instruction and as an instructional constant.

Having established that the programmer is to assign to each instruction and instructional constant - but to no other type of information - a relative index, attention must be given to the selection of these indices. In the first place, these indices must be chosen so as to group blocks of information which are likely to be relocated together. Secondly, the list of allowable indices should be as small as possible in order that the programmer may be able to remember them easily and that the machine procedure may be as simple as possible. We have found that the following groupings are most convenient:

Universal Constants (A)

These are constants which are required so generally that it would result in needless duplication to have each program carry its own set. Among these constants are the integers zero, one, and two.

Program Constants (B)

These are constants which are required for the successful execution of the program being written, but which will not, in general, be required by other programs.

Input (C)

Words of this group, or region, are read in or produced by some other program in order to provide input data for the program being written.

Output (D)

Words of this region are produced by the program being written for subsequent use by other programs.

Erasable Results (E)

These are results which are calculated and subsequently used in the execution of the program being written. No information is left in an erasable region which a subsequent program may not destroy, or erase. Thus, erasable storage is common property to all programs.

Instructions and Instructional Constants (F)

The properties of this information have already been discussed.

Since the IBM 701 deals with either 18-bit or 36-bit words, different region letters are assigned for regions containing words of the two sizes. The above letters refer to 18-bit words. The letters J, K, L, M, and N replace A, B, C, D, and E in referring to 36-bit words. All instructions and instructional constants are 18-bit words.

In addition, some instructions do not refer to storage locations. The address part of one of these instructions may specify an extent of shift or an input-output unit. Such instructions are assigned the relative index 0.

The letters listed above, A, B, C, D, etc., are used for two purposes:

1. to indicate what type of information an instruction or instructional constant refers to; and

2. to indicate to what class a word belongs.

Thus an instruction has an F indicator, or region index, associated with its location. It also has an F associated with its address part if it is to operate on another instruction. A universal constant lies in the A region. This letter A is given as a part of the location of the number, and no other index is assigned to the constant since, by definition, it may refer to no other storage location.

Consider the program referred to previously, now in relative form:

	Instruction	
Location	Op Part	Add Part
F0000	Reset Add	D0000
0001	Add	C0000
0002	Store	D0000
0003	Reset Add	F0001
0004	Add	A0001
0005	Store	F0001
0006	Transfer	F0000
Location A0001 contains the integer 1.		

Note that the indices are used both in the address parts of instructions and also in the locations of instructions and other information. The program now clearly shows that instructions F0000 through F0002 add input to previous output to form new output. Instructions F0003 through F0005 add the universal constant "1" to instruction F0001 to obtain a new instruction. Instruction F0006 returns control to the start of the sequence.

The convention has been adopted in writing programs in relative form that the first instruction of every sequence shall occupy location F0000 and that the following instructions shall occupy consecutively-numbered locations. The problem of insertion and deletion of instructions will be mentioned later. Similarly, each region associated with a program, such as the output region (D) for example, also starts at relative location 0000.

In addition, each location indicates what type of information it contains by means of a region index. Thus the assignment of these indices serves two purposes:

1. the relative indices classify instructions according to what type of information they operate upon; and

2. the regional indices classify storage locations according to what type of information they contain.

Suppose that the program is now to be assigned to actual locations. First it is punched in a card in the form shown, except that a numeric operation part is substituted for the alphabetic one. The additional information required with each instruction in regional programming requires that each instruction be assigned more than the standard storage amount prior to relocation. In operating with the IBM 701, we assign two full words, or 72 binary digits, for an unrelocated instruction, and one half word, or 18 binary digits, for an instruction in final form. In addition, it has been found useful to allow twenty card columns of explanation with each instruction. Each instruction occupies a single punched card prior to relocation. Instructions may be kept in the electrostatic storage of the 701 if desired, but it is convenient to adopt the standard procedure of storing unrelocated instructions, with comments, on magnetic tape. In this way, the programmer need not be concerned with storage limitations. Due to the practice of punching comments with instructions, binary-punched cards do not produce any substantial saving in the storage of unrelocated instructions.

After the program and associated constants have been loaded by a relocation program, origin cards are entered. These cards specify the location of the various regions. For instance, one card determines the location of the instruction region, or F region. It does this by specifying the location of F0000. The relocation program then examines the location of each word previously entered and the address part of each instruction and instructional constant for reference to the F region. Whenever this search uncovers an F indicator, the increment from the origin card is added to the four digits following the F in order to obtain the desired location or address part. This information is then stored. Similarly, a card is entered specifying the location of A0000, C0000, and D00000 and similar calculations are performed for each of these three quantities.

The information to be operated upon may appear in storage in any sequence, and the origin cards may be entered in any arbitrary order. On the IBM 701 the information being relocated is stored on magnetic tape and is brought into highspeed memory once for each origin specified. Since only one word of the information to be relocated and only one origin need be in highspeed storage at once, relocation may be performed on a program of any desired length.

The procedure outlined above may be extended in a direct and simple way to cover the problem of relocating and assembling two or more programs in such a way that these programs will operate as a unit. In this case of assembly of programs, each program must be assigned a number, or prefix, from 00 to 99 at the time it is entered into the calculator for relocation. Two programs may not be assigned the same prefix. If the programs to be assembled are being used for the first time, the prefix assignment may be made before writing begins. If library programs are being used, the convention is adopted that in library form each program refers to itself by the prefix 00. A simple substitution program may then be used to assign a new prefix.

Since a program refers to other programs by specifying appropriate prefixes, the problem of links is handled automatically during the relocation process. The instructions and constants of each program are punched in cards as before, except that now each regional index and relative index carries a two-digit prefix with it. The information from these cards is loaded into the machine (in the case of the IBM 701 it is written on magnetic tape). The necessary origin cards are then read into storage. For each region index and prefix entered, a search is made of all locations and instruction address parts previously stored. Whenever the same index and prefix is found, the number from the origin card is used as an increment to modify the four digits following the index in the program being modified. After each increment, or origin, has been treated in this way, the set of programs operated upon are completely relocated and assembled.

The choice of regions is such that the task of assigning origins is relatively simple. The region assignments may also be made so as to conserve as much storage as possible. For instance, the erasable regions of all programs may overlap; hence, the assignment of the location of E may be the same for all programs. A typical situation in assembly is the assignment of origins so that the output of one program, say program 32, is the input to another program, say program 17. This is accomplished by assigning 32D0000 to be equal to 17C0000 by means of the origin cards.

It is sometimes necessary to insert or delete instructions in a program. This may be treated as a relocation problem since it involves moving all instructions following the point at which the insertion or deletion is made. References to the instructions which are moved must be changed accordingly. The relocation procedure described above will accomplish this moving of instructions and changing of references with the addition of a simple test to its ordinary operation. This test is to determine whether a given location or address part refers to an instruction before or after the point at which an insertion or deletion has been made.

Suppose, for instance, two instructions are to be entered into the instruction sequence for program 17, and that these instructions are to follow instruction 17F0021. This means that the previous instructions occupying locations 17F0022 and following must be displaced and the references to them modified. This is done automatically by operating on instructions stored in the calculator as before. An origin card containing 17F0022 and the increment 2 is read in. Tests are then made to discover all locations and address parts in the class 17F. When such a location or address part is found, a further test is made on the four digits accompanying the indicator to determine whether the number is equal to or greater than 22. If it is, the increment 2 is added to it; otherwise no change is made. After all such modifications have been made, the two new instructions are inserted into the sequence. The same procedure is followed for deletion as for insertion, except that the instructions to be removed are taken out first. An origin card with the address of the first instruction following those deleted and a negative increment giving the number of cards removed is then entered. The remainder of the operation is automatic.

Note that the test of the four digits following the location or address part indicator may be made during the standard relocation operation. This test makes no difference in relocation since the origin is specified for the first word of a group and hence has the digits 0000 following the indicator. Every member of the group is equal to or greater than 0000 and will be relocated.

The modifications made upon instructions in the course of relocation, assembly, insertion, or deletion may be checked in the following manner. A count is made to determine how many instructions refer to each region to be used. Thus, there might be a count for the number of references to region 17F, and this count would in general differ from the counts of references to 18F or 17B, for instance. The count for each region is then multiplied by the increment to be applied in relocating the region. The resulting products are then summed. The sum thus obtained represents the total change produced by relocation of the entire set of instructions being operated upon. Thus the modification of these instructions may be checked by summing the instructions before and after relocation, obtaining the difference of the two sums, and comparing the difference with the sum of products described above.

In conclusion, we have found that Regional Programming, which is essentially the classification of each storage location according to the information it contains and of each instruction according to the type of information to which it refers, has made it possible for a machine to aid materially in the preparation of its own programs. The system was designed to aid in the operation of relocation, but it is possible to accomplish the assembly and correction of programs using this same system. The system has the further advantages that it has been found to be simple for the programmer to use and that the output of the relocation operation can be checked.

Regional Programming was developed by a group investigating possible methods of programming the IBM 701 in such a way that library programs could be reused and that programs could be written without direct reference to actual machine storage locations. Therefore, the system was designed primarily to accomplish the relocation and assembly of programs. It has been pointed out that the correction and modification of programs may also be done under this system. However, relocation by regional programming is most efficient when blocks of information can be treated as units, as is the case in the reuse of previously tested library programs. In certain types of operation, such as performance testing of a calculator, it may be necessary to modify programs a great many times by the insertion or deletion of instructions. If this latter type of operation is contemplated, the use of Symbolic Programming should be considered.

[1] The principles of the programming system proposed in this paper, particularly as they relate to the construction of instructions, are very similar to those previously developed by Dr. M. V. Wilkes and his associates in programming the EDSAC. A description of the EDSAC system may be found in "The Preparation of Programs for an Electronic Digital Computer" by M. V. Wilkes, D. J. Wheeler, and S. Gill.