HISTORICAL DOCUMENT 1958

~~COMPANY CONFIDENTIAL~~

TRANSLATION OF COMPUTER PROGRAMS

December 30, 1958

W. Buchholz
S. G. Campbell
K. W. Kaeli
M. Kloomok
E. G. Newman

Product Development Laboratory
Poughkeepsie, N. Y.

H. G. Kolsky

COMPANY CONFIDENTIAL

December 30, 1958

MEMORANDUM TO: Mr. H. T. Marcy

SUBJECT: Program Translation

The enclosed report represents the views of the Committee you appointed to consider the possibilities of the automatic translation of existing 705 programs to different languages of other computers which might succeed the 705 line, in particular, a drift transistor version of the 7070 at three to five times the speed (the "7070X"), and a lesser version of Stretch (the "7000X"). This, of course, involves the general question of whether program translation is even feasible. Our conclusions and recommendations are summarized below.

1. "Pure" simulation of one machine on another is well understood and always possible. However, simulation generally takes excessive running time on the "object" machine.

2. "Perfect" translation of a program for one machine into an equivalent program for another machine appears theoretically possible. However, the perfect translation would take an excessively long time to run on even the fastest contemplated computer. We also do not know of any hardware approach which will appreciably simplify the present translation problem.

3. An approach combining simulation and translation programs is a practical proposition subject to a number of not unreasonable restrictions. These restrictions are itemized in Section 6.2 of the enclosed report.

4. Regardless of the immediate pressure to provide a 705 replacement, there is a fundamental need to develop a translation technique. We recommend starting such a project, fully realizing that it will be a difficult pioneering effort requiring a high level of competence.

5. Our rough estimate is that a group of up to ten highly capable people should take between one and two years to complete a working translator program.

6. The efficiency of the translation approach will be considerably enhanced by the continued improvement of machine-independent languages. We recommend a continued and intensified effort in this area.

7. We conclude that the 705 TX is not a necessary addition to our line of products; that, as successors to the 705, we can and should offer the 7070X or 7000X computers with procedures for translating production programs from the 705 I, II, and II.

8. In the future, design objectives for new machines must specify means for converting the programs of all replaced machines.

The 705 TX proposition rests almost entirely on the desire to give present 705 customers a program-compatible successor. Without good translation and simulation tools this trend will inevitably continue. IBM would face a future of offering the same small group of customers an apparently endless succession of derivatives of the early 701 and 702, first in tubes, then in transistors, then in cryotrons, etc., all competing with each other and with the newer machines. We submit that this would seriously endanger IBM's continued progress. An all-out attack on automatic translation of computer programs to more advanced computers would, in our judgment, serve IBM and its customers better in the long run.

W. Buchholz

W. Buchholz, Chairman

S. G. Campbell

S. G. Campbell

K. W. Kaeli

K. W. Kaeli

M. Kloomok

M. Kloomok

E. G. Newman

E. G. Newman

pkb

Encl.

cc: C. L. Christiansen
S. W. Dunwell
J. C. Logue
H. A. Mussell

12/30/58

COMPANY CONFIDENTIAL

TRANSLATION OF COMPUTER PROGRAMS

1. Introduction

At Mr. H. T. Marcy's request, the following committee of Poughkeepsie Product Development personnel convened on December 8, 1958:

W. Buchholz, Chairman
S. G. Campbell
K. W. Kaeli
M. Kloomok
E. G. Newman

The general question posed to the committee was this: Will we have to continue to update existing machine organizations to protect present customers' investments in existing programs, in addition to developing new machines for new customers and new applications, or is there a satisfactory way to convert existing programs to the new machines? In other words, will we ever be able to break away from admittedly obsolete machine organizations? How can IBM, and thus IBM's customers, ever make progress?

More specifically, the committee was asked to consider the question of a successor to the 705 I-III machines. Is a transistorized 705 necessary, or can we offer the customer a solid state machine representative of the best we know how to build at this time?

The Committee wishes to acknowledge the advice freely given by Messrs. J. Batchelder, I. Liggett, D. L. Mordy, J. F. Parson, C. B. Poland, and J. Terlato, but the opinions expressed here are purely those of the Committee members.

2. The Problem

At the present time, there are two new solid-state computers under active development. The 7070 is intended to provide commercial customers with a machine of the approximate capability of the 705 III, but at a substantially lower cost. The Stretch Computer (Project 7000) is primarily intended to provide technical customers with a very much more powerful computer at a substantially higher cost, though the organization is sufficiently general to permit this type of machine to be applied to problems other than technical computing where the performance needed justifies the cost.

It has generally been assumed that it will be possible to bridge the wide performance and cost gaps between the 7070 and Stretch by derivatives of these machines. The 7070X, a drift transistor version of the 7070 at three to five times the speed, should provide 7070 customers who outgrow that machine with a fully program-compatible successor. The 7000X, a lesser but fully program-compatible version of Stretch, should give customers who need the powerful features of Stretch, but not its performance, a less costly machine from which they could step up to the senior version if and when needed. The 7070X and 7000X are not now under active development. They could conceivably be rather close in performance or cost or both, but their separate justification would rest on full program compatibility with their respective parent machines.

The 7070 and Stretch programs would thus appear to provide complete coverage for the near future in the 705 - 704-709 area and above. Nevertheless, Marketing has insisted that there is a strong need for additional successor machines which are program-compatible with the 705 and the 704-709 lines. The justification is that present customers have large investments in existing programs and that the advantages of the newer machines are not sufficient to justify spending large sums of money on reprogramming existing problems for new machine organizations. Thus, projects to transistorize the 709 and the 705 are now active. The intention to make these compatible but faster machines available has already been announced to the field.

There is a distinct danger of generating a multi-headed monster. We now have five different and incompatible lines of machines with wide areas of overlapping application: The 650 (with tape), the 705 I-III, the 704-709, the 7070, and Stretch. None of these lines is showing any sign of disappearing. If a new machine organization is developed in the future, it presumably becomes a sixth line. While we will win some customers from competitors and gain new customers who are not now using such machines, the market will clearly not be so large as to require these many overlapping lines. The lines must inevitably compete with each other. The sales, educational, customer engineering, and programming support for each line will add up to a staggering amount. We must also be conscious of the possibility that a competitor may find a good answer to the conversion problem, possibly by inventing an entirely new organizational approach which pays off in spite of conversion costs. Hence, IBM cannot afford to coast along without developing new techniques.

3. Will the Problem Disappear?

There appear to be two possible alternatives to the necessity of providing program-compatible successors for each line of computers IBM now produces. The first is an automatic translation of customer programs from one computer to another, regardless of dissimilarities in the organization of the two machines. The second is the development of a Machine-Independent Programming System which would permit the customer to express his problems in a language independent of the computer, present or future, on which these problems will run.

To consider the second alternative first, some progress has been made in the development of machine-independent languages. FORTRAN was a first attempt, successful only over that part of the broad spectrum of computer applications for which it was specifically designed. When modifications were incorporated to make FORTRAN more generally applicable, it was done on a basis of expediency and machine language instructions were incorporated. COMTRAN, which is not yet completed, is another attempt to develop a machine-independent language, this time for the commercial customer. Another project of this nature jointly sponsored by the ACM and their European counterparts is under way.

Although techniques have improved considerably since FORTRAN was first planned, it is too early to predict whether a truly universal language is feasible. More likely, there will be a family of machine-independent languages, each oriented towards a specific class of problems. Furthermore, there will probably be new generations for each family as programming techniques improve and computer applications will become more sophisticated. So we return to the problem of translation once again, this time from the viewpoint of converting programs written in obsolete machine-independent languages.

Applied Programming is inclined to minimize the magnitude of the translation problem by the following reasoning: 75% of the dollar invested by the customer in computer programs is for problem definition and system modification (which would presumably not be wasted in going to a new machine) and only 25% is for machine coding. It is further claimed that 40% of this 25% can be eliminated with packaged I/O programs. Thus, only 15% of the original investment is wasted in reprogramming for a new computer. This argument has convinced many 705 customers who have indicated their intention of converting to the 7070. There remain, however, customers who refuse to reprogram and insist on a program-compatible 705 replacement. The Committee feels that the reasoning advanced by Applied Programming makes a good deal of sense but this does not eliminate the desirability of automatic translation.

The feasibility of automatic translation was treated in a recent letter to the field*. The following statement attributable to Applied Programming appeared: "After considerable investigation, it has been determined that it is very nearly impossible or impractical to translate from one machine language to another machine language, or from one Autocoder language to another. Some assistance is available in the changeover process through the use of simulation programs."

The Committee disagrees. We believe that, subject to a number of not unreasonable restrictions, automatic program translation is an economically feasible proposition.

4. Simulation and Translation

A now famous theorem, proved by Turing in 1936, states categorically that the simulation of any "Turing Machine" on any other "Turing Machine" is theoretically possible. Since all of the machines we are discussing fall within the definition of Turing Machines, simulation is always a possible, though not necessarily efficient, process.

The program which precisely imitates (simulates) one machine on another is relatively easy to produce. The practical difficulty is that simulation means a great loss in performance. The time to execute the original program on the object machine, by forcing the object machine to simulate the subject machine, is very much longer than the time to execute an equivalent program written efficiently in the language of the object machine**. To justify economically the replacement of the subject machine by a simulation procedure on the object machine, the performance-to-cost ratio of the object machine should be higher by at least the same factor as the drop in performance effective during simulation.

Assuming that simulation reduces the effective performance by a factor of 10 and that cost increases as the square root of performance, two fairly reasonable assumptions, then the object machine would have to be 100 times as fast as the subject machine in order to run its programs economically. Where the performance goes up by a factor of around 2, as appears likely in going from a 705 to a 7070, then simulation would not be a good economic proposition. The senior Stretch machine will, on the other hand, have a performance great enough to make 705 simulation practical. The breakeven point is somewhere between the two machines.

* Letter to Branch Managers by C. Garrison, Jr., on "Progress Report IBM - 7070 Automatic Programs"; December 5, 1958.

** If the object machine is sufficiently faster than the subject machine, simulation on the object machine may actually be faster than running the original program directly on the subject machine. Thus, 650 simulation programs exist for the 704 which can run programs in 650 language on the 704 four to eight times faster than the 650, and more cheaply too.

Since simulation is possible and is, indeed relatively easy to do, there can be only one reason for going to the much more difficult process of translation; and that reason is efficiency. So far as translation is concerned, the following results can be proved:

A. Given any computer C, any properly defined computing job J_i, and any unambiguous definition of efficiency, there exists at least one program P_i which performs job J_i on computer C with optimum efficiency. Given two different computers, C and C', the same computing job J_i, and the same definition of efficiency (which is assumed to be machine-independent), there exists a set of optimum programs P_i on C, and a set of optimum programs P_i' on C'. Furthermore,

B. there exists at least one translator $T_k(C, C', P_i)$ which translates any of the programs P_i on C into one of the equivalent optimum programs P_i' on C'.

These theorems can be proved under conditions which are quite general and realistic. Further, for any existing computer, the translating program T can be written in a reasonable time. Since, then, it is possible and reasonable to write a translator which will translate an optimum program on machine C into an optimum program on machine C', we must say that translation is possible in the same way that simulation is possible.

But the fact that we can write a translator, which will perform an optimum translation within any reasonable definition of "optimum", does not mean that we have solved the problem; it only shows that the translation problem, like the simulation problem, does have a solution.

The difficulty with translation lies in the time of execution of the translator program itself. It is not a difficult program to write, but it is a difficult program to execute. The "perfect" translator which we know how to write would require literally years of running time, on the average, on a super-Stretch computer to translate any but the most trivial program. A special-purpose computer could be defined to execute it more efficiently, and it could be further improved by more careful theoretical considerations, but the amount of work involved would still be staggering.

The main point to be derived from theoretical considerations is that simulation and optimum translation are both possible; but both are usually too expensive: the simulator in running time on the object machine, and the translator in time required to perform the translation. A practical solution, if there is one, would have to lie somewhere intermediate between the simulator and the optimum translator. As the translated program is allowed to be less optimum, the translation times become smaller but the operating times on the object machine become larger. Eventually the program becomes a simulator, the translation time goes to zero and the operating time loss approaches the factor of 10 or more mentioned previously.

The question as to whether there is a point at which a reasonably good translation can be achieved in a reasonable amount of time depends upon the three machines in question: the subject machine, the object machine, and the machine upon which translation is performed. Considering the machines which are available for the translation (the 704 for example) it appears that considerably less than optimum translations can be obtained with reasonable translation costs. The object machine, then, could be expected to run the translated-simulated program with a decrease in performance probably greater than 2 and less than 10. This makes the chances of economical translation of 705 programs to the 7070 quite doubtful, to the 7070X better, and to the 7000X quite good.

5. Some Problems in Translating 705 Programs

The difficulty of translating at the machine language level lies in the characteristic idioms of any computer which are related to its internal structure and which have no direct counterpart in another dissimilar computer. In the 705 such idioms appear in both the data format and the instruction set.

5.1 Data Format Translation

In any 705 program there exists a marked interdependence between the data to be processed and the instructions which carry out the processing function. The data format is arranged in a manner which best suits the operational characteristics of the machine. In like manner instruction sequences occur which take advantage of the organization of the data. This implies a translation of both the data and the instructions.

The 705 enjoys the advantage that, unlike the 7070 for instance, alphabetic characters and pure decimal digits are distinguished by unique codes and are easily translated to any other code. The 705 code has the disadvantage, though, that signs of numeric fields are indicated by zone bits on the units digit which make this digit indistinguishable from an alphabetic character. Translating to machines with explicit sign coding (7070 and Stretch) requires a knowledge of the nature of the 705 instructions which operate on the field. Such knowledge is also required to change from the variable length field format of the 705 to the much more rigid format of the 7070 with its limited word packing capabilities. Although Stretch can handle variable length fields with even greater ease than the 705, it does have an upper limit on the field length which is less than the limit in the 705, so that a knowledge of the data layout is still required. The limits of maximum field length which the 7070 and Stretch machines can handle will require a change to multiple precision type of operation.

The most serious problem is probably the specification of the lengths of individual records and of groups of records. The 705 generally requires a record mark to indicate the end of a record and a group mark to indicate the end of a group. None of the object machines being considered handle records and groups quite the same way. If the records and groups have fixed lengths, analyzing the 705 program to detect the location of the respective marks will be sufficient. When the lengths are variable, it may become necessary to trace the program with sample data so as to detect the instructions which place the record and group marks into their new position.

When translating from the variable field length 705 to any fixed word length machine, it is to be expected that the new data format will require more space in memory and on tape.

5.2 Instruction Translation

Generally, each operation built into a computer, particularly of the single address variety, does only a small piece of the "macro-operation" which the programmer is trying to accomplish. The difficulty is that the intent of the programmer is not readily apparent after a macro-operation has been spread out over a series of basic machine instructions. Given a list of machine instructions, it is hard to discover the beginning and end of the set which makes up a macro-operation. This is complicated by the fact that the programmer may have interspersed the instructions making up one macro-operation with instructions belonging to another. The programmer himself is conscious of this difficulty and he frequently adds explanatory comment to guide himself. Unless these comments are in a formal symbolism which a computer program can interpret they are of little help to the automatic translation process.

Another difficulty is more peculiar to some computers, the 705 included, and it is one we would prefer to get away from. The interpretation of a single operation may depend in a major way on the data it encounters. Thus, the Add to Memory operation in the 705 performs a purely decimal addition if the first character in memory is a signed digit, but it is modified to include a special binary zone addition if the first character was an unsigned digit. It would have been better if these operations had been given separate codes. (In fact the 705 III required the inclusion of an extra code, though the old ambiguous one had to be retained to maintain compatibility with other 705's.)

Yet another difficulty arises from the fact that the programmer may make deliberate use of the incidental results of an instruction (e.g., overflow, sign of zero) which are incidental to the basic operation (e.g., Add), and may have no direct counterpart in another computer.

As a final example, the ability of the computer to modify its own instructions (which is really the foundation for the power of the stored program computer) leads to programming operations which are highly dependent on the structure of the specific computer. Instruction modification is used a great deal on the 705, which lacks index registers.

It is worth noting that a human programmer, not completely familiar with the original program, would have analogous problems in manually rewriting the program for another computer. He would have to analyze the program and guess the intent of each portion so as to formulate suitable macro-operations in his mind if not on paper. He has a substantial advantage over a computer because he can read the comments and he is better equipped to generalize from the particulars. On the other hand, the computer may have an advantage in analyzing the structure of an existing program, which can be a very laborious job.

6. A Translation Procedure and its Restrictions

6.1 Procedure

In the formulation of this report we have visualized in general terms a practical simulation-translation procedure involving the following steps:

A. Translation from 705 Machine Language to 705 Autocoder
B. Explicit Identification of Operands
C. Generalization of Machine Operations
D. Conversion to Language of New Machine
E. Debugging on New Machine
F. Conversion of Files

Most 705 programs are now written in Autocoder which is a machine-oriented language but much more useful to translation than actual 705 machine language. Instructions in Autocoder are clearly distinguishable from data, memory locations are referred to by symbolic rather than absolute addresses, and macro-operations are incorporated as desired. However, once an Autocoder program is assembled into a machine language deck, correction cards are very often inserted in machine language and the original Autocoder deck becomes obsolete. Aft er a program is debugged, the Autocoder deck is often discarded entirely. As a result, the only program certain to be correct is the one currently being run on the machine in machine language.

Step A is necessary, then, to reconstruct an Autocoder program from a machine language program. This procedure has already been proven for the 704. A 704 "disassembly" program* has been written which translates a deck of 704 binary machine language cards into the standard symbolic assembly program (SAP) language. This 704 program, although it is by no means a perfect translator, has been applied successfully to several quite large programs. This existing program is offered as partial evidence that practical translation schemes are possible.

The Autocoder language (or other symbolic language such as SAP) still contains much of the idiom of the subject machine. Step B expands the source program now in Autocoder language to identify more explicitly the characteristics of the operands and such functions as:

Numeric or alphabetic character of data.
Numbers being signed or unsigned.
Length of data fields.
Re-use of accumulator and ASU contents and indicators.
Iteration (counting) loops.
Address modification.
Program switches.
Subroutine entries.
Location of record and group marks.

* SHARE program PKDSMB, "Binary Card Disassembly Program", by A. L. Samuel and W. H. Burgin, September 9, 1956.

Step B may require a static tracing of all branches of the program or, if the data definition is not available, a dynamic tracing on the subject machine with sample data. In the course of the analysis, tables are constructed to tag each instruction and each data field with all the identifying information necessary. Any apparent contradiction detected during this analysis, for which no procedure has been specified, will be flagged and brought to the attention of the programmer, even though the program may make a simplifying assumption and proceed. At the end of the tracing, all instructions will be checked to see that none have been overlooked. Any discrepancy will be flagged.

The analysis will be simplified, and therefore less subject to error, if the data layout is explicitly provided by the programmer (assuming it is not already part of the Autocoder program). Such information may be optional or unnecessary in the simpler programs. It may or may not prove essential to successful translation of really complex programs with highly variable data layouts.

The expanded program resulting from Step B is still formulated in terms of the specific operations built into the subject machine. Step C proceeds to another and more general set of operations which can be readily converted to the object machine language. Ideally, Step C would restate the program in a truly universal programming language from which it would be easy to convert to any of many different machine languages. This would avoid the formidable task of creating a multiplicity of specific machine-pair translation programs. A good programming language equally applicable to a number of computers may well be a by-product of the development of translation programs.

If the operand identification (Step B) and the operation generalization (Step C) are completely successful, the remaining translation steps should not be too difficult. If success is not complete, subroutines will be inserted to simulate on the object machine untranslatable functions peculiar to the subject machine. These insertions will be flagged to permit manual reprogramming, if and when desired, for greater machine efficiency.

Step D, the conversion to the language of the new machine, is routine inasmuch as going from the general to the specific is a straightforward task. Difficulties could possibly be introduced by limitations of the new machine, such as insufficient memory, but insurmountable problems are not anticipated.

Given enough memory (and perhaps auxiliary storage), Steps A and B can be readily done on the subject machine, or one like it, without interfering with production runs. Steps C and D are better done on the object machine since it is presumably faster and more flexible. Possibly a third computer, one more powerful than either, could be called on for this one-time job.

Most of the debugging of the translated programs (Step E) might take place on the new machine before it is put into regular use. The conversion of the active files to the language of the new machine (Step F) and the final transition to production runs on the new machine are difficult tasks requiring careful planning so as not to disrupt production schedules. It would seem very desirable for the new machine to be able to process the files in their original format, at least temporarily, so as not to require an unproductive pass of many reels of tape.

Provision must be made in the translation procedure for the possibility that two or more separate programs to be translated may use the same data. The output of one program may be the input to one or more others, or a file may be used by more than one program. Obviously, the data layouts must match. One way to do this is to allow pre-assigned data definitions as an optional input to the translator, these definitions being supplied either by the programmer or as an output from an earlier translation.

6.2 Restrictions

The above procedure would appear to imply a number of restrictions which may be summarized here:

1. The subject program must have been debugged and must be meaningful on the subject machine.

2. The translation will not be perfect. Human assistance may be requested by the translating program. The translated program will require debugging on the new machine.

3. The translator will have to interpret idiomatic programming procedures considered standard on the subject machine, but some colloquialisms will prove unintelligible to the translator.

4. The translated program will not take full advantage of all the features of the object machine since this may require replanning of the application.

5. Since the translation will not be perfect and probably will include elements of simulation, the translated program will require more time and more memory than an equivalent program written directly for the new machine.

6. The translation process will be lengthy, and its cost must be added to the cost of doing business on the new machine.

These restrictions imply that translation will not be practical unless the object machine has a performance-to-cost ratio that is enough greater than that of the subject machine to overcome the loss in efficiency. One would, of course, expect any translator to be improved over the years, so that the efficiency should gradually be increased and the human effort reduced.

7. Conclusions

If the object machine is very much faster than the subject machine, simulation can be a practical way to make the transition to the new machine. It has the advantage that no time or manual effort is required to convert programs or files. A considerable improvement in performance can be obtained by a slight departure from pure simulation to pre-edit the original instructions and simplify their interpretation. No knowledge of the meaning of the program and data is assumed in such a pre-editing step.

At the opposite extreme, it is clearly possible to provide programming aids which would greatly simplify manual reprogramming. These might include programs to convert from 705 machine language to 705 Autocoder, to analyze the subject program for loops, to identify classes of operands, to print suitable listings, and to draw flow charts. Trial translations can be made and printed side-by-side with the original program. An advanced programming language for the object machine would facilitate writing the new program.

We feel that these programming aids can be carried far enough so that, combined with partial simulation, they become, in effect, an automatic translation procedure. The translation will not be perfect, and human assistance for specific sections of the subject program may be required. The final program may contain some errors which can be found only by debugging. However, the sum total of human effort should be very much less than with manual reprogramming, and this effort will be reduced further as the translation procedures are improved over the years.

The Committee considered possibilities of constructing special translating equipment and hardware modifications to our new computers to facilitate program translation. Translation is a sufficiently complex process, however, so that nothing short of a full-fledged computer with a large memory can begin to do the job. Nor is it evident that any changes to the new computers already under way would attack a large enough portion of the overall problem to make a great deal of difference. We do feel strongly that the translation problem should be carefully considered at the start of any new computer project.

Because of the importance of helping our customers bridge the gap to more advanced equipment, we recommend that a strong effort be started to develop translation methods. Real progress in program translation depends on getting top-notch, inventive people to work on it. A routine attempt by a large crowd may only serve to discredit translation as a valid tool. We look on this job primarily as a practical effort to accomplish a result, not to develop a theory.

It is our estimate that a group of up to ten highly competent people would be able to come up with a working translator program in a period of one to two years. We do not know whether this effort should be undertaken by Applied Programming or by Product Development. Since a theory is also badly needed, we further recommend that Research back up the effort as part of their work on general language translation. We feel that the subject deserves the best IBM has, since our future in the computer business may depend on it.

WB:SGC:KWK:MK:EGN/pkb