A SURVEY OF AUTOMATIC CODING TECHNIQUES

FOR DIGITAL COMPUTERS

by

JOHN L. JONES

B.A., Luther College
(1950)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May, 1954

Signature of Author _____
Department of Electrical Engineering, May 24, 1954

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Graduate Students

A SURVEY OF AUTOMATIC CODING TECHNIQUES

FOR DIGITAL COMPUTERS

by

JOHN L. JONES

Submitted to the Department of Electrical Engineering on May 24, 1954 in partial fulfillment of the requirements for the degree of Master of Sciences

ABSTRACT

Automatic coding techniques are attempts to reduce the amount of time and work necessary to prepare a particular problem for solution on a digital computer. This is accomplished by shifting as much as possible of the necessary clerical work onto the machine itself. Although a human must do the original analysis of the problem, a machine can be made to do the rest of the job.

There are two basic methods of approach, that of 1) compiling and 2) interpreting. There are in addition several other approaches which are not so generally applicable, the most prominent of these being the process of conversion. This technique is used mainly by binary machines.

The principal factor in determining the method chosen by a particular group for a particular machine is the storage characteristics of the machine. However, quite a number of other factors have an effect here, also.

Interest in these techniques is widespread. This interest appears to have increased the general desire for exchange of information. With such exchange, pressure mounts for a standardization of terminology. Concrete progress is being made in this direction as well as in the refinement of automatic coding techniques.

Thesis Supervisor: Charles W. Adams
Title: Assistant Professor of Electrical Engineering

ACKNOWLEDGEMENT

The writer is deeply indebted to Professor Charles W. Adams for the encouragement and guidance continually received, not only during the course of this work, but for the past two years at MIT. For this he can only offer a very sincere "thank you".

A vote of appreciation must also be given to all those people who were contacted either by letter or in person, and without whose interest and help this paper would have manifestly been impossible to prepare. Thanks also to Miss Claire Fleming who typed the final copy.

Last, but by no means least, a thank you to my wife, Nancy, whose typing of eighty-odd letters, various reports and rough drafts was no small part of the job.

TABLE OF CONTENTS

APPENDICES

CHAPTER I
INTRODUCTION

The verb "code" appears at first thought to be a relatively insignificant little word. Indeed, a person not at all familiar with the realm of computing machinery would probably be inclined to feel that the use of "code" as a verb is of infrequent occurence. However, to the person who has had the occasion to use a digital computer of any description, the verb "code" will call to mind occasions of human mistakes and subsequent lost time.

In the text at hand, a distinction between the verbs "program" and "code" will always be made. Programming a problem will indicate the process of initially taking the raw problem and analyzing it into its basic logical blocks or sections. Generally each such section is some sort of computation followed by a "yes-no" decision as to which future path of operation to follow. This procedure is often called "flow-charting"-in which a diagram is prepared indicating the flow or sequence of operations and decisions which must be made for the solution of a particular problem.

The process of coding for a digital computer is the process of explicitly expressing, in minute detail, every operation which must be performed for the solution of a particular problem. This coding must be in a symbology which is of meaning to the machine in question, and in general a symbol is needed for each operation. Such operations or instructions most usually consist of one arithmetic operation such as adding, subtracting, etc., or one logical operation such as a transferral of information from one location to another, or one jump, based on a yes-no decision, from one set of operations to another. The difficulty that begins to appear is one of trying to write down every arithmetical and logical operation in the solution of a particular problem. Every contingency must be prepared for and in general not one instruction of any sort may be forgotten if the correct answer is to result. Further complicating matters is the fact that the human mind does many of these simple operations automatically.

This fact "helps" one forget essential instructions.

Even in the light of the foregoing discussion, forgotten steps are generally not the most common mistakes. The machine also has to be told exactly where the data it is working with may be found and where the instructions are located. For this purpose, the machine has a storage unit with numbered addresses. Upon being told only the number of the address desired, the machine can go and obtain the information. The mistakes arise here in the human's either remembering incorrectly where a piece of information is stored or else thinking of 36 and writing 63 or some such. Since most machines do not distinguish between numbers and instructions, this may result in a piece of data getting into the control circuits or operating on an instruction as a number. When this happens, almost anything may result. Often the machine goes into a "loop" - that is repeats an unending cycle of instructions over and over until stopped manually.

All of this would be bad enough in itself - but in addition, a particular routine may need to be recopied for reasons of legibility or sequencing of operations. Or perhaps the coder remembers, or finds in checking, a forgotten instruction. That instruction must then be inserted and all subsequent addresses modified accordingly. If any instruction after the insertion is operated on by another instruction, the address in the operating instruction must be modified to correspond to the location change of the instruction to be operated on. This means more forgotten changes.

In the end, the finished code is often checked instruction by instruction by another coder to detect any obvious mistakes. Unfortunately another coder may check a couple of pages, finding no mistake, and then begin to feel "Well, this guy's pretty good, I'll just hurry along - " and about this time the checker misses a mistake, and so it goes.

The reason these mistakes in codes are so disgusting, (other than the damage caused to the coder's ego) is the fact that the machine will find these clerical mistakes and operate incorrectly or not at all. Since time on most large scale machines is valued at several

hundred dollars per hour, the people who pay the bills are loath to have a coder hold up production to find (using the machine) such a mistake, and, being under pressure, the coder may very well miss the mistake again and again.

The problem should now be clear - how to "speed-up" the process of coding a particular problem by eliminating as many sources of mistakes as possible. The ideal situation would be to get rid of the coding altogether. This is also desirable from the point of view that coding is downright boring and laborious work, once the first flush of "glamour" wears off. This latter fact results from coding being, basically, a clerical type of job. It is a matter of remembering a group of addresses and instructions and putting them correctly in the right place.

Fortunately, we have the best clerk in the world available, a digital computer. Not only does this clerk work hundreds of times faster and makes no complaints to a twenty-four hour a day, seven day week, but it seldom malfunctions in doing so (provided we have given the proper instructions to the machine in the first place!).

In other words, we want to automatize the coding procedure to the point where the machine can do most of the coding itself. These methods are popularly called automatic coding techniques.

Perhaps the reader has noticed the careful avoidance of the word "error". In the interest of promoting a standardization of terminology, the following statement[1] is offered:

> "Numerical analysis has <u>errors</u>; programs, coding, data transcription, and operating have <u>mistakes</u>; a computer has <u>malfunctions</u>."

There is another pressing cause for automatizing methods of problem preparation. This is the so-called "one-shot" type of operation.

1. <u>A Programmer's Glossary</u>, Dr. Grace M. Hopper, 1 May 1954

By this we mean problems that are done only once. We must consider two phases of the problem: 1) the one-shot use and 2) the one-shot user. The one-shot use apparently occurs most frequently in engineering and scientific applications. The preparation and debugging of a routine which will solve the problem may take days, weeks, or even months, and then the problem is run on the machine in a matter of hours and the whole routine is junked when the answers are once obtained. Unless some method of hurrying up the acquisition of the final checked routine is evolved either a large staff of professional programmers and coders is needed or production will suffer. Since the machine is accurate as well as fast the solution is apparent - automatize the method and let the machine do the rest.

Consider next the one-shot user. Not only does he generally have a one-shot problem but rarely does he have more than a passing acquaintance with the methods of problem preparation for a digital computer. In general he should prefer to prepare his own problem because:

1) he doesn't want to spend a lot of time explaining to someone all the facets of a problem, some phases of which he may have a "feel" for that would be hard if not impossible to convey.
2) he knows no one else has as much interest in his problem as he does.
3) he doesn't want to wait an indefinite but often lengthy time for a programmer to become available.
4) he wants to spend as little money as possible.

Here we have a paradox - on one hand a man who knows little if anything about computers and on the other that same man with a problem he desires to solve using a computer. The solution? Of course. Automatize the method so it is simple for the man to learn and let the machine do the rest.

What tasks can be relegated to the machine? Certainly not the initial operation of analyzing the problem and resolving it into

its logical parts. This requires the power of reason which is (at the moment at least) the human's exclusive possession. Here we are fortunate - for the ability and training required to do such an analysis and chart the flow of possible events is no birthright of the programmer. It is an essential requirement in the executive, scientist and engineer alike. Once the problem is analyzed, which must be done in any case or we do not know what the problem is, the machine can do the rest of the entire job - that is, if a pseudo-code can be provided so that each of the basic, logical steps of any problem can be expressed in one, two or a few lines of coding. This pseudo-code should resemble as closely as possible the everyday working symbology of the individual with the problem. This latter aspect will be discussed more extensively in the final chapter.

Historically, the idea of mechanizing the process of problem preparation is not recent. Charles Babbage was on the track of a "library of subroutines" although he did not call it that. He states:[2]

> "There are therefore two sets of cards, the first to direct the nature of the operations to be performed - these are called operation cards: the other to direct the particular variables on which those cards are required to operate - these latter are called variable cards..... Under this arrangement, when any formula is required to be computed, a set of operation cards must be strung together, which contain the series of operations in the order in which they occur..... Every set of cards made for any formula will at any future time recalculate that formula with whatever constants may be required. Thus the Analytical Engine will possess a library of its own."

For the purpose of this paper, a library of subroutines will be defined as any set of routines or subroutines which are designed to perform a defined function or operation. These libraries as used

2. Babbage's Calculating Engines, Henry P. Babbage, E. and F.N. Spon, London, 1889, Chapter VIII, pages 159, 160

today may contain a great number of different routines. The content of the library varies considerably from installation to installation depending upon the particular types of problems met and the mode in which the library is used.

At this point apologies must be made to any readers who are not at all familiar with the principles and terminology of computers. Up to this point an effort was made to define the problems and their motivations in terms that could be understood by non-computer people. However, from this point on, it will be assumed that the reader has at least a speaking acquaintance with the computer field.

CHAPTER II
THE BASIC FORMS OF AUTOMATIC CODING

2.1 Definitions

In this chapter, two rather straightforward examples are given illustrating the two basic methods of automatic coding commonly employed. By "straightforward" it is meant to be implied that the examples will demonstrate the definition of the method as closely as possible. The fact that most techniques either contemplated or in use are combinations of varying amounts of the two basic concepts do not make the selection of examples a simple choice.

An example of a system using only a library of subroutines (with or without automatic address modification) will not be given since the method is covered rather completely in "The Preparation of Programs for Electronic Digital Computers" by Wilkes, Wheeler & Gill. However, it should be noted that such libraries are the basic building block of these two automatic coding schemes.

The generally accepted names for the two methods are 1) compiler and 2) interpreter. The rather loose usage of these two terms causes considerable confusion. Many groups call their method either a compiler or an interpretive technique when in actuality they use some combination of the two. Therefore, we define the terms as follows:[1]

> Compile (verb) - The process of producing from pseudo-code a specific routine for a particular problem by:
>
> 1) decoding elements of information expressed in pseudo-code and segmenting the problem;
> 2) selecting or generating the required sub-routines;
> 3) transforming the subroutines into specific coding and entering them as elements in the problem routine;
> 4) maintaining a record of the subroutines used and their position in the problem routine.
>
> Compilation decodes the pseudo-code and processes static and dynamic subroutines and generators to

1. A Programmer's Glossary, Dr. Grace M. Hopper, 1 May 1954

produce a specific routine for a problem before computation.

Interpret (verb) - To produce the desired solution of a particular problem by:

1) decoding an element of information expressed in pseudo-code;
2) selecting the required subroutine;
3) carrying out the required operation by means of the subroutine;
4) continuing to the next element of information.

Interpretation decodes the pseudo-code and refers to static subroutines during computation. It does not produce a specific routine for a problem.

2.2 The Compiler

For the example of a compiler technique we will use the so-called "A-2" compiler as conceived and prepared for UNIVAC by Dr. Grace M. Hopper and her staff at Remington-Rand's Eckert-Mauchly Division. This method, the third to be realized by Dr. Hopper's group, has been checked and in use since mid-November 1953. The first attempt at a compiler, "A-0", was completed before May 1, 1952. "A-2" shows a considerable amount of refinement based on experience gained in working with "A-0" and "A-1".

Information is prepared for the compiler in a pseudo-code. This pseudo-code mnemonically indicates the function of the code word and is a three-address code. In general, each pseudo-instruction calls for a subroutine. Since the basic UNIVAC word is twelve alphanumeric decimal digits, the operation and each address has three decimal digits assigned to it so that one pseudo-instruction is one UNIVAC word.

To begin a compilation, the machine is furnished with:

1) the compiling routine tape,
2) the information tape (pseudo-code definition of problem),
3) the library of subroutines tape,
4) the blank tape for the compiled or "running" program.

It should be noted that a UNIVAC is an alphanumeric binary coded decimal machine equipped with six to ten magnetic tape units which read and write (simultaneously if desired) at the approximate

rate of 10,000 alphanumeric characters per second. The machine does not wait for this process to end before resuming computation. This is about the best input-output equipment presently available.

One of the basic elements of the compiler is the library of subroutines. For the "A-2", this library is a "floating-point" library with the usual arithmetic, trigonometric, root extraction, logarithmeric and exponential functions available, as well as input, output and control transfer routines. There are several other routines called "generative" routines. Dr. Hopper, in her "Glossary"[2], defines generate as:

> Generate (verb) - To produce coding by assembling and modifying primitive elements such as parameters and skeletal coding.

When a generative routine is called in, certain arguments and control words must be furnished in addition to the original pseudo-code instruction. Control is then transferred to the generative routine and this routine actually makes up the coding necessary to the specifications in the aforementioned arguments.

When the generative routine has finished, control is transferred back to the compiler routine which places the generated subroutine on the running tape.

To allow for peculiarities in some problems, the programmer is allowed to specify "OWN CODE" and then immediately follow with the necessary instructions in UNIVAC code which will go "as is" onto the problem tape. If the "OWN CODE" is lengthy, it may be put on a tape as a "special" library routine and called in by the compiling routine when needed. Generally, only two or three lines of "OWN CODE" are necessary.

The compiling routine is read into storage where it immediately assumes command by transferring some control information to the running tape. This information that may be needed by the

2. A Programmer's Glossary, Dr. Grace M. Hopper, 1 May 1954

compiled or running tape for changing from one memory load to another, transfers of control from one memory load to another, automatic overflow routines, constants, etc. Then a 60 word block (all UNIVAC magnetic tape input-output is in 60 word blocks) of pseudo-code is read into storage and the consecutive processing of each code word is begun.

As stated previously, in general each pseudo-instruction corresponds to some subroutine in the library. Each subroutine has a "call-word" and the library is ordered in ascending values where $(1 < 2 \ldots < 9 < A < B \ldots < Z)$. The compiler looks at the operation indicated, finds the corresponding subroutine, reads it into storage, makes a record of the location of the first line of the subroutine in the running program, and then transfers the subroutine word-by-word to the output block. During this word-by-word transfer all necessary modifications are made. These modifications include the three addresses of the pseudo-instruction which are relative to the beginning of the working storage block (s). When the entire subroutine has been transferred, the next pseudo-instruction is called up from storage and the process repeated. If the same subroutine is called for more than once it is recompiled each time with the exception of the four floating-point arithmetic functions, which are "frozen" in the internal storage element. As the output block becomes full, it is written on the running tape and a new block is built up.

Should the final running program exceed the storage space allocated by the compiler routine in making up the running tape storage layout of 560 words (= 1120 UNIVAC instructions), the running program is automatically "segmented". If an iterative loop exists and there is a possibility that this loop may enclose parts of two segments if assigned by the machine (thus causing a complete storage change twice per iteration), the programmer may simply state "SEGMENT" at each end of the loop and the machine will accordingly segment at those points.

The end result of all this (assuming no mistakes in the pseudo-code) is a tape in machine code which needs no "debugging" and is ready to be run. The compiler tape and library of subroutines tape are no longer needed.

Since the pseudo-code version of the problem may be written directly from the flow chart many of the pitfalls as discussed in Chapter I are skirted. The pseudo-code description of a problem is considerably shorter than the machine code routine and more easily found if committed. To give an idea of the order of magnitude of the reduction in number of instructions, the following information was compiled from an "A-2" manual[3].

An optical ray tracing problem was solved with "A-2". The problem was to pass each of twelve rays through seven surfaces. The initial rays were specified by three coordinates and three direction numbers, the surfaces by their radius of curvature. The index of refraction on each side of the surface and the distance between surfaces was given. The pseudo-code took 100 instructions, the final program took approximately 1000 machine instructions.

2.3 The Interpreter

To pick an example of the "interpretive" mode is more difficult. Few groups use a "pure form" of the interpretive method but rather they add features which particularly suit their needs or are easily attained with their machine. However, the system designed for International Business Machines' 701 general purpose digital computer is about the best example of a straightforward interpretive method. This system is called "SPEEDCODE I".

Each pseudo-instruction is made up of one three-address and one one-address operation. The single-address operation is used for conditional and unconditional transfer of control, address modifications and error checking operations.

3. The A-2 Compiler Systems Operation Manual, Remington-Rand, Inc., 15 Nov. 1953

The three-address operation section has the usual arithmetic, trigonometric, root extraction, exponential and logarithmic functions available for floating-point computation as well as quite a large number of information transfer and input-output instructions.

The address modification scheme is a programmed "B-line" type of operation. Any one, any combination of two, or all three addresses of the first operation may be modified simultaneously.

The pseudo-code may be considered to be mnemonic, however, the 701 is not alphanumeric and if the original code is written in mnemonic code, the punch cards (which are the principal form of input) must be processed on standard IBM sorting and gang punching equipment. This processing will cause the instruction deck to be punched with the proper numerical code.

This deck is then read by the 701 and converted to binary (the 701 is a binary machine) at the rate of 150 cards per minute. One card has either one SPEEDCODE instruction or five data words.

The machine then starts the problem solution by calling up a pseudo-instruction (now in binary form), deciphering it, and then performing first the three-address operation and then the one-address operation. When this instruction has been completely executed, the next instruction is called up and it is deciphered and executed as above. This process continues in sequence or as directed by transfer of control instructions until the machine is instructed to print out the final results and stop.

The reduction in the number of instructions necessary to define a problem in pseudo-code over the number required by a hand tailored code should be about the same as in the compiler example, since these two pseudo-codes are very similar.

It is noteworthy that both codes are three-address. Many groups are choosing such schemes because the three-address code seems to be less succeptable to coding mistakes.

The reader is referred to Appendix A for a short discussion of two other quite different approaches using an interpretive technique.

CHAPTER III

A SURVEY OF PRESENT METHODS

3.1 Other Approaches

There are several approaches to automatic coding, other than the two basic forms, which should be discussed. While these methods are no less powerful perhaps, they are techniques which particularly utilize the peculiarities of a certain machine to the extent that the method is not generally applicable to all machines.

The first of these is the process called "conversion". In general, binary machines require some conversion process to occur in order that the input information may be made intelligible to the machine. This is true unless the input information is already in binary form. However, few people are willing to write their programs in binary, especially non-professional computer users. Consequently, a system is often provided whereby the routine may be written in alphanumeric characters using decimal numbers. It is the responsibility of the machine to do the necessary conversion to binary. This is essentially an automatic coding technique - shifting some clerical work onto the machine for the convenience of the programmer.

Many other features may be provided at the same time with no great amount of extra time used. These include the usual relative and symbolic addressing facilities. If, as is often the case, the conversion program is to be followed by one of the other two basic methods, selection of subroutines for floating-point work may be done during the conversion. Very often conversion is so intermixed with another of the methods that it is virtually impossible to determine where one ends and the other begins.

Another technique is commonly called "assembly". This is basically a compiler approach in that a routine of this nature assembles and modifies subroutines as they are called from the library. The method often allows for symbolic addresses. Usually no "generative" type subroutines are found in these methods, but this is about the

only difference. Quite a large number of the IBM 701 installations have developed routines along this line, as well as some other groups.

The so-called "generators" constitute a third approach. These routines are used mostly by UNIVAC installations and usually on either output editing or sorting problems. In general, a "background block", a format, or other specifications are given along with a skeletal form of the coding to be used. This skeletal coding very often is in a "packed" form and the function of the generator is to unpack this coding and supply suitable addresses. The entire machine code routine is then generated. It is apparent that these routines are restricted in the sense that they are presently applicable only to very specialized problems.

Several groups are approaching the problem by using a pseudo-code made up of both "real and abstract" instructions. This implies that the code uses actual machine instructions as well as programmed instructions. The routine may or may not require notification when a change from real to abstract instructions, or the reverse, is made. Such an approach allows the routine to be considerably more flexible for use by the professional programmer. Savings of time are effected by doing at least part of the routine at ordinary machine speeds, which are generally ten to fifty times greater than the abstract section of the routine.

This idea particularly lends itself to the new Naval Ordnance Research Computer (NORC) being built for the Navy by IBM. The NORC is a three-address floating-point machine. It is therefore unnecessary to program floating-point arithmetic but it is still desirable to use subroutines for the usual mathematical functions. Therefore, the use of both real and abstract instructions in the compiling routine being devised is the answer to the problem.

Some effort may be noted in the line of preparing two or more techniques which use the same pseudo-code. Considerable work has been done by the Computer Control Company, Inc. of Point Mugu,

California, on such a project. The ultimate goal of this group is to develop methods whereby an interpretive routine will be used to check the coding and then the checked pseudo-code will be given to a compiler routine for final preparation of the machine code. If possible, it is extremely desirable to have more than one technique available, since then the technique most suitable for the particular problem may be chosen.

In addition, virtually all combinations of these methods may be found so as to form a continuous "spectrum" of techniques. One of the best examples of such a combination is the Comprehensive System in use at MIT's Digital Computer Laboratory. Since Whirlwind I is a binary machine, and input is in the form of alphanumeric binary-coded decimal punched paper tape, a conversion program is necessary. This routine also provides for "floating" and relative addresses and selects the proper programmed arithmetic subroutines to allow floating-point operation. When the conversion is completed, a binary tape of the converted problem routine may be punched out if desired and necessary subroutines are then compiled from a library recorded on magnetic tape, including an interpetive routine to perform the actual problem solution.

3.2 Factors Involved in Choosing a Technique

It is apparent that the choice of a particular method of automatic coding definitely is not based on the inherent virtues or lacks of the two basic forms or any of the other approaches mentioned. Rather, it is the characteristics of the machine involved which are the major forces in guiding one group in one direction and another group some other way. Even more precisely, the amount and availability of storage seems to have the biggest single effect. However, modifying effects are attributable to problem types encountered, alphanumeric qualities of the machine, automatic checking features, serial or parallel operation, personal likes or dislikes of the programmer preparing the method, etc.

In order to attempt a compiler or assembly operation, a large amount of relatively high speed storage is a "must". This may be either magnetic tapes or drums. This is even more important if the compiled routine is "unravelled" (sometimes called subprogramming), which implies that the compiled routine has no iterative loops. Every instruction is written down before starting computation and no later modification of instructions is performed. In certain cases this may allow a considerable time saving, but fast input-output equipment (to high speed storages) is necessary since the program becomes very long.

Other factors that tend toward compiler or assembly routines:

1) alphanumeric and automatic checking features,
2) minimal access coding requirements,
3) repetitive type problems.

On the other hand, interpretive techniques are often found with:

1) parallel binary machines with little or no automatic checking,
2) machines with slow input-output equipment,
3) one-shot type operation.

Most research type problems, for which only one solution is desired, lend themselves well to an interpretive technique. Problems that will need to be solved a number of times may more effectively be handled by a compiler since the compilation results in a specific routine and re-assembly is not needed for each solution.

It is noteworthy that there are a few groups who have studied the virtues of automatic coding and decided not to adopt any such technique. It seems that in these cases the type of problem encountered is the major factor. These are the installations which have problems of large size which must be recomputed (with different input data) many, many times. The general feeling is that such a problem is best solved by hand tailoring the most efficient code possible.

At this point it would be well to state that problem type largely determines how much any particular method is used, although the generality of the method has some effect. Some installations (a surprisingly large number in fact) use their automatic coding methods on virtually every problem they have to solve. Very often, however, these installations do not have problem types that vary a great deal and therefore they have designed a technique which in detail specifically suits their needs. On the other hand, some installations use their technique on only a portion of their problems. These are generally groups that meet a wide variety of problems, some of which are of a highly repetitive nature.

In any event, one may find installations that use no automatic coding, other installations that use almost nothing but automatic coding methods, and practically all degrees of usage between these two extremes.

3.3 "Post Mortem" Techniques

Considerable work is being done on routines which aid in the location of mistakes in the problem routine. These are called "post mortems" because they attempt to diagnose the trouble from which the routine "died". They may operate either automatically or on demand. In general they provide the programmer with information which should be helpful in locating the mistake. This information generally includes:

1) location in the program at which the mistake was detected,
2) contents of the accumulator and other registers,
3) the contents of the storage locations selected by the programmer,

and may include:

1) a record of the last few control transfers that were effective,
2) present contents of any storage locations that have been changed since the start of the program (replacing feature 3 above).

(For an example of this type of operation see the Summer Session Computer under MIT in Appendix B.)

Unfortunately, many methods that provide highly desirable information, if adapted into an automatic coding technique would slow down computer operation an intolerable amount. This results from the fact that many methods require that a record be kept of all storage changes, transfers or control, etc. which occur. This in turn means that each instruction must be individually checked and the record accordingly changed or not changed. For this reason a compromise is usually necessary. This often results in the adoption of a "tracing" routine. This routine is a "dynamic" technique, which is called into use only when a program is known to have mistakes. The routine then proceeds through the entire program, step-by-step, typing out selected information along the way. This information may be typed out after completion of each instruction or only upon finding certain instructions. When the problem routine is "debugged", the tracing routine is completely removed from use and the problem runs at the usual speed with no record kept for mistake location purposes.

CHAPTER IV

A SURVEY OF CONTEMPLATED TECHNIQUES WITH COMMENTS

4.1 Terminology

From the very outset of this survey, it was evident that one of the biggest problems was going to be that of terminology. Not only does virtually every machine type have its own language, but some discrepancies are found even among users of the same machine type.

Take for example, the type of address assignment scheme (allowed by most autocoding techniques) whereby the person preparing the code or pseudo-code for the machine designates blocks or regions of coding by symbols (generally alphanumeric), which may have some particular significance to that person. It is the duty of the machine to assign definite or fixed values to these addresses. They may be thought of as "floating" addresses by the programmer, who doesn't generally know or care at the time the routine is written what these values will be. But besides being called "floating" addresses, they obviously may be called with equal accuracy "block", "regional" or "symbolic" addresses.

Fortunately, more and more people are becoming acutely aware of this problem. Instead of arguments with no ending, compromises are being effected. Many people are coming to the viewpoint that it is high time to adopt an attitude of cooperation and show respect for the concepts of others. In this way a standardization of terminology may be realized in the near future and this will promote the exchange of ideas and experience. This in turn eliminates duplication of work and thereby saves time which is all too valuable for the professional programmer.

4.2 Universal Codes

An element that many believe to be both a cause and result of the pressing need for a standard terminology is the interest in

and desire for a "Universal Code". Interest was expressed by many groups and actual work by some people has begun. Ideas along this line were about as follows:

1) the code would be a "symbolic logic" mechanized for computing purposes.

2) the code would be written directly from a "flow chart" of the problem - in fact the flow chart would be the code.

3) the person preparing a problem would write the program in Universal Code, theorectically at least, not knowing what machine would be used. He would be aided in this by a "dictionary" completely describing the code and its standard subroutines (also in Universal Code).

4) each machine would have its own personal "translator" which would prepare the machine code problem (a compiling technique).

These ideas are largely those of Dr. Saul Gorn of the Ballistics Research Laboratory at the Aberdeen Proving Grounds. Dr. Gorn has done a considerable amount of work along this line of thought. He emphasizes that the great advantage of such a scheme lies in the fact that the "flow charting" of a problem is an ability and training that is an absolute necessity for any good engineer, executive or scientist regardless of whether or not a computer is involved. Equipped with the knowledge of some conventional symbolic logic and a dictionary on universal coding, these people could easily prepare their own problems for computation merely by analyzing them, which they certainly do anyway.

It is understandable that Dr. Gorn should arrive at this point of view. The Aberdeen Proving Ground is a multi-machine installation with three large-scale machines, namely ENIAC, EDVAC and ORDVAC. The prospect of learning three machine codes (they also have a Bell Relay Computer and IBM equipment) is no particular pleasure. Also, a programmer is usually only interested in how soon he can use a machine, which may be any one of several if a Universal Code is used.

4.3 Multi-machine Installations

In a multi-machine establishment the interesting question arises, "Is it feasible to use these machines in either parallel (for checking) or serial (for speeding computation)?" The Aberdeen group has tried both approaches with notable success. Perhaps more interesting than the parallel operation (which is actually built into some machines) is the serial use. They have used all three machines in a sort of "assembly-line" problem solution, each machine doing a phase of the problem for which it was particularly suited (or at least suited as well as the others). The final output on punched cards was then taken to the IBM section for printing.

This bit of information certainly lends credence to the long standing opinion of Dr. Hopper that an effort should be made to build small sized computers with very general and elementary instruction codes. The inference is that the person who needs a large installation could operate as many units as necessary in serial. Ultimately, the object is for the computers to directly control each other. The whole system could then be programmed to appear to be a large scale machine with any or all facilities desired. The "programmed hardware" (e.g. "B-lines", multi-address codes, etc.) cannot fail because it does not actually exist.

This aspect should at least be investigated because of the interesting possibilities. For example, suppose these units could be produced at a reasonable price and there were plenty of units available since they should admit to being rapidly as well as inexpensively built. The small user could then operate parallel units for checking. The large user, while he may not be able to afford a duplicate set, could almost undoubtedly afford at least one spare unit. Instead of replacing a faulty component or chassis of components, he may now replace an entire computer. This should result in a decrease of time lost due to machine failure even over the present day "unitized" machines with adequate spares. From past experience with computer maintenance, the advantage of having removable

units and available spares is recognized by most computer people.

4.4 The Analytical Differentiator

While such operation is probably far in the future, attention should be called to some work that points in this direction. This is the method developed by Mr. Harry Kahrimanian[1] for analytical differentiation with a digital computer. Mr. Kahrimanian submitted this technique for a Master's thesis at Temple University while an employee of Remington-Rand.

The method analytically computes any or all of the derivatives from one to ninety-nine of most of the elementary functions and any finite combinations thereof. There are twenty-four elementary functions given and with all finite combinations of these possible, most cases may be obtained.

The desired function (made up of the twenty-four) is stated in symbolic form, along with the derivative or derivatives desired, and the machine does the rest. The output is the analytical expression for the derivative in the same symbolic form, although not necessarily the "simplest" mathematical form. Reducing the derivative at each step to its simplest form is not necessary (nor easy!) for the computer.

Some examples of the work of the routine (taken from the thesis):

1) Determine the first eight derivatives of:

$$y = a \sin^3 (b x^4)$$

Computer time used: 17 seconds

2) Determine the 15th order derivative of the function:

$$y = (x + 1)^3 \sin (2x + 1)^2$$

Computer time used: 23 seconds

1. Analytical Differentiation by a Digital Computer, Harry S. Kahrimanian, May 1953

3) Given: (e = epsilon)

$$\frac{dm}{dq} = a \cos \phi \, (1 - e^2 \sin^2 \phi)^{-\frac{1}{2}}$$

$$\frac{d\phi}{dq} = \cos \phi \, (1 + e^{12} \cos^2 \phi)$$

$$\frac{d\phi}{dm} = a^{-1} \, (1 - e^2 \sin^2 \phi)^{\frac{1}{2}} \, (1 + e^{12} \cos^2 \phi)$$

Determine: $\frac{d^2m}{dq^2}$ through $\frac{d^{15}m}{dq^{15}}$

and: $\frac{d^2q}{dm^2}$ through $\frac{d^{15}q}{dm^{15}}$

Computer time used: 4 minutes, 25 seconds

This is one step farther than our present concepts of automatic coding - that is, the computer taking on itself the duty of performing mathematical functions, not just arithmetic, and giving the answers in a symbolic form. The output of the Kahrimanian differentiation is in a pseudo-code which is usable by the "A-2" compiler. The compiler takes this pseudo-code output, compiles the machine code and thus gives the final form for numerical evaluation.

4.5 The Form of Pseudo-codes

As a matter of fact, the pseudo-code output of the differentiator goes through a stage of "translation" before the compilation actually begins. This is interesting because it brings up the possibility of having several pseudo-codes, adapted for various type problems, where each such code would have a "translator" which would put the information into a form usable by the compiler routine.

The pseudo-code for use in (say) accounting problems would resemble as closely as possible the format that the accountant actually uses in his everyday work. The pseudo-code for the scientist or engineer would resemble the equations actually used in common

notation. The pseudo-code for the executive would be in terms particularly suited to his needs.

This is fundamentally the motivating concept of the "algebraic" approach - to make the coding so simple that very little has to be rewritten and few if any new methods or notations learned. Most programmers feel that things are heading this way. This does not mean toward the interpretive algebraic methods, but toward the idea of having a notation for coding consistent with everyday usage.

Professor Charles W. Adams of MIT points out that two problems immediately confront us when an algebraic type code is formulated. First, mathematical notation is not entirely unambiguous in the sense that one problem may often be written several ways, or conversly,a set of equations may sometimes be interpreted several ways. Secondly, and by no means insignificant, is the mechanical problem that arises - the typewriters and similar equipment that is used in preparing codes just doesn't have the necessary symbols available. In other words we are shackled by the design of a relatively insignificant (compared to a computer) piece of auxiliary machinery. However, it seems that modifications of this equipment is prohibitively expensive and there cannot be much hope for a change in this situation for some time to come.

In any event, this idea of having a different pseudo-code for each problem type is really the dual of Dr. Gorn's approach. Dr. Gorn suggests one code for many machines, this latter idea suggests many codes for one machine.

These ideas are not incompatible if we think of the one machine having many codes as the 'universal" machine. That is, there would be many pseudo-codes, each particularly adapted to a certain type of problem, and each pseudo-code would have a translator associated with it which would translate that pseudo-code to the Universal Code of the non-existant universal machine. The problem could then be solved by any machine which had a translator that could express the Universal Code in its own machine code. As a matter of fact,

this procedure is exactly that followed by the differentiator, although the "Universal Code" involved is the pseudo-code for the A-2 compiler.

The attitude that a programmer takes on this matter depends pretty much upon whether he has one problem which he may solve on anyone of several machines or several problems and only one machine available.

4.6 Two Interesting Trends in the Computer Field

Two very interesting lines of thinking, which appear to be of recent origin but already have widespread acceptance are these:

1) a willingness to share ideas and work with other (and often rival) computer groups;
2) the opinion that the internal speed of a machine is not too critical.

A good indication of this new thinking, as indicated in 1) above, is the library of information accumulated in the process of preparing this survey. The project was tackled with some reservation and apprehension which may be attributed to the fact that an inquisitive graduate student writing a thesis does not have a very strong argument for requesting information that may be considered "personal property" by a computer group. However, the collection graduated from a drawer to a shelf and wound up as a two foot (plus) stack on a chair with enough more reports promised to nearly double the stack. People who had no reports available at the time, often wrote long letters giving some of the details of their work. Needless to say, this was greatly appreciated. The large number of symposiums, conferences and seminars on the subject also indicates this general trend.

This also adds to the desire and needs of a standardization of terminology. A standard notation is essential for communication of any sort and from this need will come greater efforts to resolve the difference. A great deal of work along this line has been done by Dr. Hopper and a committee of the Association for Computing Machinery.

As for the second trend, whether a problem takes eight machine hours or ten machine hours to run does not seem to make much difference if the problem preparation time can be cut from months to weeks or even days. Most programmers agree with this in reference to problems that are not highly repetitive. As previously stated, most programmers agree that a hand tailored routine is still the best for problems that will be done over and over.

Dr. Hopper believes, and has some examples which show, that the result of a compiling technique should be a routine just as efficient as a hand tailored routine. Some others do not completely agree with this. They feel the machine-made routine can approach (and possibly very closely in most cases) hand tailored coding, but they believe there are "tricks of the trade" that apply to various special cases that a computer cannot be expected to utilize.

Again, this business of obtaining a very efficient resulting code is not at all critical except for highly repetitive problems. It is doubtful that a production chief would ever complain over a few hours of machine time if perhaps several weeks or months of over-all solution time were saved on most problems.

4.7 Conclusions

To use a rather trite phrase, "automatic coding is here to stay". This seems obvious, not only from the progress that has been made, but also from the great amount of work planned for the immediate future and the widespread interest in these techniques. Not much information was obtained as to how many people are interested in this work even though they don't have a computer. However, virtually every group that does have a machine of some sort has at least thought about the problem.

The time is certainly "ripe" for great strides to be made in standardizing terminology and thereby opening the way for an exchange of information. This will in turn lead to better and better automatic coding techniques by eliminating much duplication of work

and permitting incorporation of ideas which are best adapted to the machine involved.

It is sincerely hoped that this paper may in some small way aid in promoting these ideals.

APPENDIX A

ADDITIONAL APPLICATIONS OF THE INTERPRETIVE METHOD

Two quite different approaches to the general problem of coding were investigated to some degree.

The first of these interpretive methods is the so-called "algebraic" approach. By this we mean methods in which the information (equations and data) are given to the machine in a form which approximates the mathematical notation as closely as possible. This means that once the problem (the class of problems is somewhat restricted) is formulated on paper in algebraic notation it is pratically coded for the machine.

Reports on two versions of this approach were available. One of these was prepared for Whirlwind I by Dr. J. H. Laning of the Massachusetts Institute of Technology's Instrumentation Laboratory and the other for UNIVAC by Mr. J. Robert Loagan of Remington-Rand's Eckert-Mauchly division. Although Dr. Laning's technique uses notation which more closely approaches usual mathematical notation than Mr. Logan's method, we will use Mr. Logan's work as descriptive example. The reason for this is based on the fact that UNIVAC is an alphanumeric binary-coded decimal machine and the technique applied is truly an interpretive one per the definition given in Chapter II. Since Whirlwind I is a binary machine and input is by paper tape on which is punched a Flexowriter[1] code version of the problem, there must mecessarily be a Flexo-to-binary conversion, during which some of the work required by the pseudo-code is carried out. This does not fulfil our definition of a "pure" interpretive method.

Mr. Logan's technique is called "Short Code" due to the abbreviated length of the problem information input necessary. By virtue of UNIVAC's direct Supervisory Control typewriter input and output (as well as high speed magnetic tape) the method is extremely flexible. Since the Short Code is a floating-point routine primarily

1. Flexowriter is the trade name of a standard line of secretarial equipment.

intended for use as a mathematical research tool, it is desirable to be able to change coefficients, equations and constants in the middle of a computation. With the Short Code no "machine code" input tape needs to be reworked, the new information may be typed directly into storage by the Supervisory Control typewriter.

Original information may be typed directly into the machine for short problems or prepared on magnetic tape in the case of more lengthy problems. Answers may be obtained either on magnetic tape or on the Supervisory Control typewriter at the option of the programmer.

The problem data is in the form of floating-point numbers. The instructions are in the form of two-decimal-digit "packets". The twelve decimal digit UNIVAC word can accomodate six such packets, and as many words as necessary may be strung together to state the equation. Each word is deciphered in packets from right to left. Each packet may represent an alphanumeric variable or an operation. Eighty variables are in the machine at a given time (although more may be on tape) and these are denoted by one letter (S through Z) and one number (0 through 9). A variable must be assigned a value before it is used in any equation. This means the method of solution must be explicitly stated.

The operations available are the arithmetic functions, integral root and power routines, logarithmic, trigonometric, exponential and logical operations. In addition, optional stops or "breakpoints" and input-output instructions are provided. Designations for "equal" signs and parenthesis are given.

Although the Short Code is not particularly mnemonic or in mathematical notation, nonetheless, equations are written in the form of equations. For example, if one wished to evaluate

$$a = (b + c) \sin d$$

(assuming b, c and d to be previously assigned values and letting a = s0, b = s0, c = s2, d = s3)

he would write (in two UNIVAC words):

```
sl  07  s2  02  60  s3
00  00  00  s0  03  09
```

Writing the algebraic equation above the Short Code symbols:

```
a   =   (   b   +   c   )   sin  d
s0  03  09  sl  07  s2  02  60   s3
```

These comments should explain the points of question:

1) the "equals" sign assigns the value computed prior (to the right) to the equal sign to the variable immediately following it.

2) a single packet of "00" tells the machine to "skip" the rest of that word. No operation may follow the equal sign or a "skip".

Dr. Laning's method is essentially the same with these noteworthy differences:

1) The equation used as an example for the short code would be written by Dr. Laning as:

$$a = (b + c)\ F^2 d$$

2) Dr. Laning provides a list of twenty-three functions (denoted F^1, F^2, ... F^{23}) which include the regular and inverse trigonometric, square root, exponential, logarithmic and hyperbolic functions. Also provided is a method for solution of ordinary differential equations by the method of Gill, which is a variation of the fourth order Runge-Kutta technique. An nth order system must first be reduced to n first order equation. d/dt is represented by D.

3) Allowance is made for 250 variables in high-speed storage and an easy method is provided for assignment and recall of variables to the drum. Any lower case letter without any subscript from 0 to 1023 may be used.

4) No change can be made in the middle of computation since Whirlwind I does not have a Supervisory Control input typewriter. Output is on magnetic tape which may later be printed.

Both methods have some automatic error diagnosing aides.

The second rather different use of interpretive techniques is the approach of Mr. James E. Kelley, Jr. of George Washington University's Logistics Research Project.[2]

The Logistics Research Project Computer is a special purpose machine designed to perform simple operations on large masses of data. It is a binary coded decimal machine, using the binary excess-three code. The machine is controlled by a 40 program step plugboard. This board allows about twenty different operations which include the arithmetic operations (except division) and a variety of control and rearrangement operations. All operations are performed in five internal high-speed registers. Storage consists of a drum with over 14,000 twelve-digit storages available. Information is entered on or extracted from the drum with either teletype equipment or magnetic tape.

Mr. Kelley devised several methods based on interpretive techniques which make this special purpose machine appear to be a general purpose machine. In so doing he removes two very serious limitations from the special purpose machine, namely:

1) removal of the 40 program step per program restriction,
2) removal of a plugboard restriction which allows only five branching operations. This removal makes any number of independent sequences available.

To accomplish this, Mr. Kelley designed a plugboard (after choosing suitable conventions) which selects and interprets parameters stored on the drum. The parameters in turn direct the plugboard to perform specified operations on other drum stored data.

2. Information taken from a paper, "Extensions of Programming for the Logistics Computer", by Mr. Kelley; prepared for publication in the June 1954 issue of Logistics Papers.

The instructions (or parameters) on the drum are four address and sequencing is consecutive (by the plugboard). Three operands and a result are used since the machine considers the arithmetic operation to be a compound multiplication and addition. That is, considering the four addresses to be u, v, y and x:

in general	$uy + v = x$
for addition	$(1)y + v = x$
for subtraction	$(-1)y + v = x$
for multiplication	$uy + 0 = x$

For logical operations:

u = address of next order if v contains a positive number

v = address of antecedent

y = operand { shifted right retaining sign if $v > 0$ / shifted right cyclically if $v \leq 0$

x = result

Two options are available for output instructions (via punch).

Mr. Kelley mentioned in a letter that almost all of his work has been in the nature of research and therefore problems solved have been limited to those for testing the various plugboard designs. Mr. Kelley feels however that these techniques may be applied in such a way to certain problems that solutions may be obtained for problems which otherwise would be impossible (or at least very difficult) to solve otherwise with the equipment available to him.

Several efforts along this same line have been made for the IBM CPC. However, the only concrete information (other than Mr. Kelley) has come from Mr. T. M. Bellan of McDonnell Aircraft in St. Louis and Mr. Rex Rice, Jr. of Northrop Aircraft in California. (see Appendix B). It should be stated that this particular aspect was not carefully investigated due to the great amount of information obtained for large-scale machines.

APPENDIX B

COMPUTER GROUPS INTERESTED IN AUTOMATIC CODING

It is felt that a complete list of all the people that have been contacted in regard to this thesis, along with a brief description of the work in progress by each group, will be of interest to most computer people. This feeling arises from the fact that it was believed upon undertaking the thesis that at most probably a dozen groups would be involved and only seven were actually known. However, by the time this appendix was compiled, a total of fifty seven groups had been written and in addition personal contact with another half dozen groups was made. Even then, all the possibilities were by no means exhausted.

This list is offered in the hope that it will promote correspondence between the various groups and thereby effect a sharing of ideas and information. With this in mind, comments are made where pertinent as to the availability of literature. Those groups which presently have no reports available are in general willing to discuss their work by letter or personal contact.

The list is alphabetical by name of the group involved. The individual names given are those of the people with whom conversation and/or correspondence was carried on. All government agencies are listed under United States Government. Descriptions are abbreviated to keep the list short enough to be readable. All techniques are "floating-point" unless otherwise stated.

Apologies are offered to those groups that should have been contacted and were not due to their interests being brought to my attention too late for correspondece or else not at all.

In the event that any group has been misrepresented in this list, owing to faulty interpretation of their literature or correspondence, sincere apologies are offered. Every effort was made to avoid such mistakes but it is reasonable to assume some listings will be incorrect.

1) Bell Telephone Laboratories — Mr. R.W. Hamming
Murray Hill, New Jersey

No machine at present, interested in IBM 650.

Mr. Hamming is interested in a "from the top down" approach to automatic coding. This sounds like an algebraic approach. He states that the use of the method should be "natural".

2) Boeing Airplane Company — Mr. R.E. Porter
Seattle 14, Washington — Physical Research Unit

IBM 701 - Large library of "general purpose" routines, many of which are routines for solving problems or parts of problems peculiar to Boeing. These are arranged in a final program by an "assembly" routine. Output (of routine) is on either magnetic tape or cards. No reports presently available.

3) Burroughs Research Laboratory — Mr. Alex Orden, Manager
Paoli, Pennsylvania

Burroughs Laboratory Computer, intermediate speed with drum for main storage unit. Using 1) relative address modifier, 2) automatic step-by-step type-out routine for program checking, 3) a generative type of routine which sets up an "unravelled" program (no iterative loops) for matrix multiplication to save computer time.

4) Columbia University — Mr. Joseph A. Scott
632 West 125th Street — Electronic Research Lab.
New York 27, New York

No work in this field at present or contemplated.

5) Cons. Vultee Aircraft Corp. — Mr. Ben Ferber
San Diego 12, California — Engineering Computation Lab.

Expecting delivery of ERA 1103 this summer. Auto code is being prepared for this. No information yet published.

6) Curtiss-Wright Corporation — Mr. Russell W. Everett
Wood-Ridge, New Jersey — Public Relations

No literature available.

7) Computer Control Co., Inc. — Mr. Russell C. McGee
Point Mugu, California

Raytheon RAYDAC, large-scale general purpose, serial memory and control units, parallel arithmetic unit, binary (36 bit word), four

address, automatic checking by weighted counts, magnetic tape input-output. Developing and using both interpreters and compilers using the same pseudo-code with object of using interpreter for code checking and then using checked code to instruct compiler. Some reports available.

8) Computer Research Corp.
3348 West E. Segundo Blvd.
Hawthorne, California
Dr. A.D. Hestenes
Director, Applications Dept.

Has initiated studies of automatic coding techniques, however, no material suitable for distribution at this time.

9) Consolidated Engineering Corp.
See ElectroData Corp. (affiliate of CEC)

10) Douglas Aircraft Company, Inc.
Santa Monica, California
Mr. John Lowe

IBM 701 - Has developed an assembly program which translates symbolic to actual coding. The use of the word "assembly" by Mr. Lowe suggests a type of compiler technique since some other IBM 701 groups perform a compiler type of operation which they call "assembly". No reports available at present.

11) ElectroData Corporation
717 North Lake Avenue
Pasadena 6, California
Mr. Kenneth L. Austin, Super.
Liason & Instruction Section
Technical Service Dept.

ElectroData Computer (formerly CEC Computer), decimal machine, 10 decimal digits plus sign per word, single address, series-parallel operation (i.e. digits are parallel, words are serial), 4000 word drum storage plus an 80 word "quick access" section (0.85 ms. as compared with 8.5 ms. in regular storage section) internal checking, "B-line" type operation, paper tape input-output. Has done considerable work on interpretive systems, two have been codes, checked and in use. One system (IDA) is a conventional interpretive technique, the other (DOTI) is a faster more versatile technique developed to utilize some of the unique operating features of the computer. Literature describing these systems not yet published.

12) Eckert-Mauchly
(See Remington-Rand)

13) Engineering Research Associates
(See Remington-Rand)

14) Ferranti Limited
Moston
Manchester 10, England

Mr. Eric K. Robertson
Computer Sales Department

FERRANTI MARK I* (a later model of the MARK I), serial, binary (20 or 40 bit word may be used), electrostatic storage of 10,000 bits, drum storage of 650,000 bits, "B-line" operation, paper tape input-output. Straight machine code uses only the 32 teletype characters so that both addresses and instructions are written in teletype code. This gives a base 32 number system and the least significant digit is to the left. Ferranti is developing a simplified mnemonic code using decimal addresses and B-line. Not yet published. Considerable work had been done in past with library of subroutines technique.

15) General Electric
Post Office Box 196
Cincinnati 15, Ohio

Mr. H.R.J. Grosch
or
Mr. Donald L. Schell
Aircraft Gas Turbine Div.

IBM 701. Two systems developed by this group - GEPURS and SEESAW. The writer believes these are interpretive techniques. GEPURS is a 3 address decimal system designed for use by a novice. Has usual arithmetic, trigonometic, logarithmic, exponential and logical functions. For the "professional" programming group at G.E. a more flexible system (SEESAW) was devised, single address decimal system, programmer may go back and forth from pseudo-code to machine code. Pseudo-code has all of regular arithmetic, logical and other functions and some structure as regular machine code. Use of floating or symbolic addresses and relative addresses allowed. Published literature will soon be available.

16) General Electric
920 Western Avenue
West Lynn 3, Mass.

Mr. Allen Keller
Medium Steam Turbine Dept.

IBM 701. Has devised several noteworthy routines. One is a conversion routine which converts alphanumeric coding (either actual machine code or pseudo-code) and decimal numbers to binary form and punches out a binary deck. The other routine, LT-2, is an "assembly" routine which makes up the final program from either real machine code or the pseudo-code. All problem routines are made up and checked for use with LT-2. Since the machine used is not at Lynn, all input data is telegraphed to the machine, where the proper pre-coded program is selected and run using LT-2. Output is wired back to Lynn. This technique is feasible because problems are of limited types. Very good write-ups available.

17) George Washington University — Mr. James E. Kelley
707 22nd St. N.W. — Logistics Research Project
Washington, D.C.

Detailed description of Mr. Kelley's approach in last half of Appendix A. Reports available.

18) The Glenn L. Martin Co. — Mr. Kenneth D. Engle
Baltimore 3, Md. — Sales Department

Machines used only on specific problems all of which bear a security classification. No literature available.

19) Harvard University — Mr. John Harr
Cambridge, Mass. — Harvard Computation Lab.

Not much being done in this line. They do use a "coding box" which facilitates punching of paper tapes by automatically introducing subroutines etc. This is a different approach to the problem.

20) Hughes Aircraft Co. — Dr. E.C. Nelson, Head
Culver City, California — Advanced Electronics Lab.
Hughes Research & Dev. Lab.

It is understood that Hughes is building their own machine. A little work on autocodes has been done but no reports available.

21) The Institute for Advanced Study — Dr. John von Neumann
Princeton, New Jersey — or
Dr. H.H. Goldstine
School of Mathematics

IAS machine. No systematic studies have been done, however, several special codes have been developed. Literature soon to be available.

22) International Business Machines — Dr. G. Truman Hunter
20 E. 57th St. — or
New York, New York — Mr. Harlan Herrick
Applied Science Dept.

Detailed description of Speedcode I for IBM 701 in Chapter II. In addition, some work has been done on "assembly" routines and I understand, from other sources, a study of an algebraic approach is underway. Reports on Speedcode presently out of print but will be available shortly.

23) The Jacobs Instrument Co. — Mr. Donald H. Jacobs
Bethesda 14, Maryland — President & Technical Director

JAINCOMP series. No information has been published on subject.

24) McDonnell Aircraft Corp.
Post Office Box 516
St. Louis 3, Mo.

Mr. T.M. Bellan, Supervisor
Dept. of Applied Math.

Two IBM CPC's, basic operation controlled by plugboard, actual problem solution controlled by deck of coded program cards. Have tried two different techniques, one with some success. This involves wiring a set of plugboards which will provide the means for developing the program deck. The method is limited to simple coding changes. No reports available.

25) Mass. Institute of Technology
Cambridge, Mass.

Prof. Charles W. Adams
Digital Computer Lab.

Whirlwind I, binary (16 bit word), entirely parallel, magnetic core, drum and magnetic tape storage, paper tape input, output on scope which may be photographed automatically or magnetic tape which is printed later.

Two systems have been developed: 1) The Comprehensive System provides the same single address operations as the regular machine code except that these are floating-point operations. Two Whirlwind words (32 bits) are used as the basic word length and the programmer may pick any number of bits for the exponent, that is, he initially chooses a (30-j,j) number system with (30-j) bits in the mantissa and j bits in the exponent. $j = 6$ is the commonest choice giving a range of approximately 10^{-19} to 10^{+19}. Alphanumeric address assignments (floating address), and relative addresses may be used, a programmed cycle counter operation is provided. Programmer may go back and forth between pseudo-code and machine code. Program is typed on Flexowriter equipment and a conversion is necessary to straight binary. Machine does this (all routines and subroutines are on the drum) and at the same time selects the proper sets of arithmetic routines according to the (30-j,j) number system asked for, also assigning all alphanumeric addresses. If asked for, the machine will give a "binary tape" of the converted program. The machine then begins the problem solution, interpreting an instruction and performing it, selecting the next instruction etc. The Comprehensive System is used for virtually all problems solved on Whirlwind I. Reports soon available.

2) For teaching purposes, the so-called "Summer Session Computer" was designed. This is a programmed floating-point, general purpose computer, designed so that most human coding errors will be detected by the machine. The single address code is in every way designed to be mnemonic and the computer is decimal as far as the user is concerned.

The machine discriminates between instructions and numbers for purposes of mistake detection. Alphanumeric addresses may be used and a cycle counter function is provided. "Remainder" and "excess"

registers for division and multiplication are provided with logical operations possible on the contents of these. A very large number of other logical operations are provided.

In case the machine detects a mistake in coding such as finding an instruction in the arithmetic unit or a number in the control unit, it stops and gives a "post mortem" which includes:

1) location of present instruction which caused mistake to be noticed,
2) contents of accumulator and any other address specified by the instruction in (1) above,
3) last ten "jump" or transfer of control instructions which were performed,
4) contents of every memory location which has been changed since the program started.

This system is used only by students at MIT who are in introductory computer courses.

The program for the Summer Session Computer is on magnetic tape. It may be called into Whirlwind I by merely pressing a button. The machine thereupon becomes the Summer Session Computer and information that is read on the paper tape reader is converted and interpreted by the SSC routine. Reports available.

26) Mass. Institute of Technology — Dr. J.H. Laning
Cambridge, Mass. — Instrumentation Lab.

Whirlwind I. See Appendix A for discussion of Dr. Laning's algebraic approach. Excellent report available.

27) Monroe Calculating Machine Co. — Mr. I.M. Gardoff
Morris Plains, N.J. — Applications Research
Monrobot Laboratory

Monrobot machines (at least two sizes). The larger machine has unusually large drum capacity and numerous tape units. This group is engaged in a study of automatic coding techniques, no reports yet available. Mr. Gardoff states, "The smaller Monrobots utilize a four address system and numerical techniques, which together, yield a procedure not too unlike common algebra. The larger calculators are somewhat similar."

28) Northrop Aircraft, Inc. — Mr. Rex Rice, Jr.
Hawthorne, California — Asst. Chief of Computer Serv.

IBM CPC's and "a new machine of advanced design that is on a restricted contract". Have done a considerable amount of work on autocodes for the CPC but no reports published. Effectively this type

of autocoding is a method whereby only data is fed from cards and the sequence of operations is on a plugboard.

29) Oak Ridge Nat'l Lab. — Dr. A.S. Householder
Post Office Box P
Oak Ridge, Tennessee

ORACLE. No work to report at this time. The writer understands that there is some interest, however.

30) The Rand Corporation — Mr. Wesley S. Melahn
1700 Main Street
Santa Monica, California

IBM 701. No written report yet available, however, Mr. Melahn describes an "assembly" program that they have been using which appears to be a compiler type of operation. The pseudo-code consists of mnemonic alphabetic symbols with decimal and/or alphabetic addresses. The machine does as much of the clerical work as possible including incorporation of library routines into the final program.

31) RCA Victor — Mr. John H. Waite, Jr.
Cooper Street — Bldg. 13, Floor 2
Camden, N.J. — Section 598

RCA's new machine has not yet been announced and consequently no literature on their programming techniques has been released. I believe they are very interested in this type of work.

32) Raytheon — Dr. R.F. Clippinger
Waltham, Mass.

No reports available at present, not much work being done other than a relative address modifying routine.

33) Remington-Rand, Inc. — Dr. Grace M. Hopper
1624 Locust Street — Systems Engineer
Philadelphia 3, Pa. — Eckert-Mauchly Division

Compiler technique described in Chapter II. Excellent reports available.

34) Remington-Rand, Inc. — Mr. J. Robert Logan
23rd and Allegheny — Eckert-Mauchly Division
Philadelphia, Pa.

Mr. Logan's algebraic approach is described in Appendix A. Excellent report available.

35) Remington-Rand, Inc.
507 18th St., South
Arlington, Va.

Dr. G.F. Cramer
ERA Division

ERA 1101, binary (24 bit word), serial, single address with an address modifying operation using manual switches, 16,384 word storage drum, binary point fixed at right end, paper tape input, typewriter and/or paper tape output. Have developed a subroutine library in minimum access coding, which contains the usual function, and which takes at most two or three (in a few cases) drum revolutions. To use the library, the subroutine is written on the drum by the machine, addresses modified according to specifications and then punched back out for manual assembly.

ERA 1103, binary (36 bit word), parallel, two address, same arithmetic as 1101, 1024 words of electrostatic storage, 16,384 words of drum storage, instructions may use either ES or drum, four magnetic tape units provided, various options on input-output equipment. Work is progressing on a pseudo-code which is a combination of instructions to be interpreted and actual machine instructions. The actual machine instructions used consist mainly in the logical operations. The library of subroutines will be on a magnetic tape and the machine will assemble and modify these routines along with the actual machine instructions into final running program on the drum. At the end of this process, the program is transferred to ES and the routine begun. No reports presently available.

36) Stanford Research Institute
Stanford, California

Mr. William H. Kautz
Research Engineer

Mr. Kautz has written a paper on the work done at Stanford Research Institute entitled "Optimized Data Encoding for Digital Computers". This paper will appear in the IRE Convention Record, Part 4, 1954 which is to be published in June.

37) Underwood Corporation
35-10 36th Avenue
Long Island City 6, N.Y.

Mr. Samuel Lubkin, Director
Electronics Computer Div.

ELECOM. Are doing very little with automatic coding techniques and have no published information.

38) United Aircraft Corp.
East Hartford 8, Conn.

Mr. Walter A. Ramshaw
Research Department

IBM 701. Mr. Ramshaw's group is mainly interested in the use of the Speedcode I technique and many improvements are contemplated. It is believed no reports are presently available.

UNITED STATES GOVERNMENT

39) Hdqrs. USAF, Pentagon — Mr. Joseph Natrella
Washington, D.C. — AFAPA - 3b

UNIVAC (Air Force Conptroller's Office). Not much work has been done due to the nature of the problems to be solved. These are problems that must be solved over and over with different data and it is felt that hand-tailored coding pays dividends. A former employee, wrote an interesting paper on the application of the theory of sets to compiling routine but no other work has been attempted. Copies of this paper are available.

40) Wright Air Development Ctr. — Dr. Clarence Ross
Wright-Patterson Air Force Base — Aeronautical Research Lab.
Dayton, Ohio

OARAC and awaiting an ERA 1103. Some work has been done on a library of subroutines and its use of OARAC, however a large effort is contemplated for the 1103.

41) US Army Ordnance Proving Ground — Dr. Saul Gorn
Aberdeen, Md. — Ballistics Research Lab.

ORDVAC, EDVAC, ENIAC. Dr. Gorn has been working on a "Universal Code". This is discussed in some detail in Chapter III. No reports publically available.

42) Army Map Service — Miss Nora Moser
Washington, D.C.

UNIVAC. Improving and using a compiler very similiar to Dr. Hopper's "A-2" but more adapted to Map Service problems. No report available at present.

43) New York University — Mr. Roy Goldfinger
253 Greene Street — Institute of Math. Science
New York 3, N.Y.

UNIVAC (Atomic Energy Commission). Mr. Goldfinger has developed a version of a compiler which is particularly adapted to the problems met by his group. These problems are mostly of mathematical physics equations. Mr. Goldfinger allows the programmer more flexibility in arranging the location and size of storage assignments. He has also developed several "service" routines for the compiler for addition to the library, etc. Good write-up available. This compiler is used by virually all the programming staff at N.Y.U. Excellent report available.

44) U. of California Radiation Lab. Mr. Thomas W. Wilder, III
P.O. Box 808
Livermore, California

UNIVAC (Atomic Energy Commission). Many problems encountered by this group run for hundreds of hours and these are hand-tailored. For smaller problems they use a three address technique (called "3-D Codes") which has the usual arithmetic, trigonometric, exponential and logarithmic functions and the logical operations. This is a compiling technique. Also have done considerable work on Editing Generators. Reports soon available.

45) Bureau of Census Mr. Don Hieser
Washington, D.C.

UNIVAC. Due to specialized problems, almost no work has been done. However, a rather specialized version of a compiler was tried on a problem in which it was required to prepare 105 tables including 8300 households obtaining column and row totals, as well as certain percentages and medians. Mr. Hieser stated that several hundred hours of coding and debugging time were saved. He also indicated that up to this time the main emphasis has been on obtaining as much computer time as possible, now, however, the interest is changing to autocoding. No published reports.

46) National Bureau of Standards Mr. Joseph H. Wegstein
Washington, D.C. or
Mr. Charles J. Swift
Bldg. 83, SEAC Lab.

SEAC. Presently in use is the so-called "Base 00" routine which allows all the usual floating-point as well as some conversion operations. Four address code, essentially a compiler or assembly technique. Work was started on "universal" code (called Basic Code I) for natural science problems. Three alphanumeric addresses with two additional addresses available for transfers. Project was suspended due to lack of a sponser and other higher priority work. Some reports available.

47) David Taylor Model Basin Mrs. Betty Holberton
Washington 7, D.C.

UNIVAC. This Navy group as yet is not using any form of automatic coding. Mrs. Holberton is presently investigating the field to determine the best approach for the particular problems they meet. She is doing considerable work on "rerun" procedures, that is, methods by which a problem may be restarted in the middle of a computation should some machine failure occur which would void results from the

point of the failure onward. She contemplates building two compilers, one for mathematical problems, one for logistics problems. Has done considerable work on edit generators in past.

48) US Naval Proving Ground
Dahlgren, Va.

Dr. E.K. Ritter, Director
Caomputation & Ballistics

Awaiting the NORC. No details of this machine being built by IBM for the Navy, have been released as yet. However, it is a very high speed (well over 10,000 operations per second), floating-point, 3-address machine, and is equipped with extremely fast magnetic tape units (upwards of 60,000 decimal characters per second). The compiler being devised by Mr. Gene H. Gleissner and Mr. Karl Kozarsky of this Department utilizes all the regular machine instructions (which are floating-point operations) as well as a large subroutine library which includes all the usual functions as well as regular and inverse interpolation and integration formulas. The library routines are called by single instructions. Symbolic addresses are allowed and the instructions need not be ordered. A subroutine is not recompiled each time it is called for and at the end of the compilation the problem solution begins immediately. No reports available at present.

49) University of Cambridge
Free School Lane
Cambridge, England

Mr. Eric N. Mutch
U. Mathematical Lab.

Contemplating development of a routine such as the Comprehensive System of MIT. Has done considerable work on the library of subroutines idea. No reports available at present.

50) University of Illinois
Urban, Illinois

Dr. Stanley Gill
168 Engineering Research Lab.

ILLIAC, binary machine, sexadecimal arithmetic. Not too much work has been done on automatic coding other than an extensive library of subroutines including floating-point interpreters. "Everyday coding" is based on a routine of Wheeler's very similar to the Initial Orders in "The Preparation of Programs" by Wilkes, Wheeler and Gill. This routine takes decimal and relative addresses and simplifies the use of the library by providing modification techniques. Some reports available.

51) University of Michigan
Ypsilanti, Michigan

Mr. James H. Brown
Willow Run Research Center

MIDAC, binary (45 bit word), serial, three address code, 512 word acoustic storage, 6144 word drum storage with provision for three

additional drums, input by high-speed paper tape to be supplemented by magnetic tape equipment, output by typewriter to be supplemented by scope and high-speed mechanical or photographic printer. This group has developed an input conversion and computation system called MAGIC somewhat similar to MIT Whirlwind I Comprehensive System. Floating, relative and decimal address are allowed on input program as well as subroutines. Reports available.

52) University of Toronto
Toronto, Canada

Miss B.H. Worsley
Computation Center
McLennan Lab.

FERUT (Ferranti machine). It is understood that work is being done on a decimal system input scheme. An extensive library of subroutines has been built up for the use of the programming staff.

53) Wayne University
Detroit 1, Michigan

Mr. Wesley C. Dixon
Computer Lab.

UDEC, single address, decimal machine, 10 digits per word, drum storage, input by teletype equipment or photoelectric tape reader, output by typewriter or paper tape. Machine is still in a stage of development and consequently not much work has been done on auto-codes. Are using a routine to convert from floating or symbolic address to fixed address. Some reports available.

In addition to this list, the following groups were written with no reply received. This is most likely due to either addressing the letter to the wrong person or else mailing the letter of inquiry too late to allow the group involved to reply. Included with each group is a statement regarding what is known about the work being undertaken or heard from other sources.

1) Bell Aircraft Corp.
Post Office Box 1
Fort Worth, Texas

Nothing known about this group.

2) Bendix Aviation Corp.
Hawthorne, California

Nothing known about this group.

3) Carnegie Institute of Technology
Pittsburgh, Pa.

Nothing known about this group.

4) General Motors
Detroit, Michigan

Getting an IBM 701. Nothing else known.

5) Grumman Aircraft Corp.
Bethpage
Long Island, N.Y.

Nothing known about this group.

6) Lockheed Aircraft Corp.
Burbank, California

IBM 701, second 701 on order. This group has developed a floating-point technique called FLOP which is believed to be similar to the other 701 techniques. Also has developed a technique which allows matrix operations to be performed directly which is floating-point.

7) Los Alamos Scientific Lab. (AEC)
Los Alamos, New Mexico

IBM 701. Has developed two interpretive systems, a single address technique called DUAL and a three-address method called SHACO. It is understood these are similar to other 701 techniques. Regional addressing and assembly programs have been developed and considerable work is being done on "post-mortem" and other mistake analysis routines.

8) North American Aircraft, Inc.
Los Angeles International Airport
Los Angeles 45, California

IBM 701. Has developed a single address assembly program with the usual addressing features.

To make the list somewhat more complete, the following possibilities are submitted for additional information. These were discovered too late to permit correspondence.

Consolidated Vultee Aircraft Corporation
Fort Worth Division
Fort Worth, Texas

National Security Agency Washington, D.C.	Mrs. Dorothy T. Blum Technical Consultant
US NOTS, Math. Division China Lake, California	Mr. Harley E. Tillitt, Head Computing Branch