fer may merely be to the next command in sequence.

Class N-5, Test Transfer Operations. There are three operations in this class, one for each index register. They function in the same manner that the TN1 operations do, except that the transfer to the command in α is nullified if the limit tally for that register (which has been carried in a negative form) becomes equal to or greater than zero. The program then proceeds to the next command in sequence, and the use of all functions of that register is denied. The operations are:

Written		Signifying
TX1	α, k	Augment all registers using R1 by k and transfer to the command in α unless the limit tally for R1 equals or is arithmetically greater than zero
TX2	α, k	Same as above, except refer to R2
TX3	α, k	Same as above, except refer to R3

Appendix I

Some typical programming for matrix operations is given here to illustrate the use of PRINT I instructions. An example is given for the multiplication of a 6 by 5 matrix by a 5 by 4 matrix to produce a 6 by 4 matrix. This operation is symbolized as:

$(M1ij)(M2jk) \rightarrow (M3ik)$

and for convenience in following the indexing the complete matrices are given as:

$$\begin{vmatrix} M111 & M112 & M113 & M114 & M115 \\ M121 & M122 & M123 & M124 & M125 \\ M131 & M132 & M133 & M134 & M135 \\ M141 & M142 & M143 & M144 & M145 \\ M151 & M152 & M153 & M154 & M155 \\ M161 & M162 & M163 & M164 & M165 \end{vmatrix} \times \begin{vmatrix} M211 & M212 & M213 & M214 \\ M221 & M222 & M223 & M224 \\ M231 & M232 & M233 & M234 \\ M241 & M242 & M243 & M244 \\ M251 & M252 & M253 & M254 \end{vmatrix}$$

$$= \begin{vmatrix} M311 & M312 & M313 & M314 \\ M321 & M322 & M323 & M324 \\ M331 & M332 & M333 & M334 \\ M341 & M342 & M343 & M344 \\ M351 & M352 & M353 & M354 \\ M361 & M362 & M363 & M364 \end{vmatrix}$$

For simplicity in explanation the elements have been assigned regional addresses symbolic of their row-column identification in the matrices. In actual operation they are usually loaded row by row in sequential addresses, but the same indexing principles apply without exception.

The programmer would write the following set of instructions:

Address	Op	Variable Field	Comment
B001	CS2	60	Set R2 to 0, limit to 60
B002	CS3	4	Set R3 to 0, (or reset)
B003	RWR	5, 1, 10, 0	Row-column
B004	MAD	M111, 2, M211, 3, M311, 2+3	product summation
B005	TX3	B003, 1	Step up R3 by 1
B006	TX2	B002, 10	Step up R2 by 10

Note that the development was chosen so that the elements of the result matrix were computed row-wise. Additional instructions could be inserted to print the matrix row-by-row during computation.

It is an easy matter to abridge the foregoing program for matrix-vector multiplication. A good practice problem is the coding of:

$(M1ij)\{V1j1\} \rightarrow \{V2i1\}$

Appendix II

Fig. 1 demonstrates the effect of the RPT command on timing. Average operation

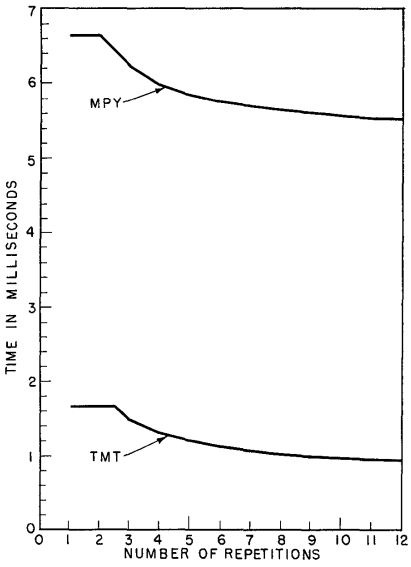

Fig. 1

times for a repeated series are shown versus number of repetitions.

MPY, when performed only once, takes 6.63 ms. When repeated, the first MPY takes 8.57 ms, the last takes only 4.72 ms, and all intermediates take 5.33 ms. The break-even point is at two successive MPYs.

TMT, when performed only once, takes 1.65 ms. When repeated, the first TMT takes 3.20 ms, the last takes only 0.46 ms, and all intermediates take 0.76 ms. The break-even point is between two and three successive TMTs.

Times for the first operation of a repeated command are high because they include the time for the RPT and its interpretation. It is obvious that a considerable time-saving may be effected by repetitive operations. They should be considered carefully not only for obvious operations such as matrix manipulation and block transfer, but also for linearly progressive calculation. Here it is advisable to group, by judicious choice of addresses, operations of the same type to take advantage of the reduced timing.

The IBM Type 705 Autocoder

ROY GOLDFINGER

THE *705* Autocoder is a system of automatic coding for use with the International Business Machines Corporation (IBM) EDPM* type *705*. However, before description of the system, the distinction between automatic programming and automatic coding will be emphasized. This will help to place the Autocoder (which, of course, is "automatic coder") in its proper perspective.

Programming is concerned with the general organization of a computer process. It encompasses such considerations as selecting the proper array of peripheral components, organizing the high-speed and auxiliary memories for the most economical access to information, deciding whether to process in one pass or two, and so on. These are programming problems which must be faced before any computer code can be written profitably.

Coding, on the other hand, is concerned mainly with a single command or an item of information. The job of the coder is to translate from the general pattern laid

ROY GOLDFINGER is with the International Business Machines Corporation, Poughkeepsie, N. Y.

* Electronic data-processing machine.

down by the programmer to the series of elemental machine-language commands necessary to operate the computer.

A human coder is capable of being both programmer and coder at once. That is, he may take the programmer's statement of an event to be performed, decide upon the best way of accomplishing it, choose among several alternatives, introduce considerations such as scaling, checking, detection of logical as well as machine errors, and simultaneously reduce the whole to the necessary series of machine-language entries. The machine coder, or "automatic coding system," generally falls far short of the ability of the human coder in the decision-making and optimizing area, while it outstrips him by far in speed, accuracy, and the production of legible results.

The finest of the so-called "automatic programming systems" are, in fact, excellent automatic coders. Those programs which have attempted to proceed further into the programming realm, intent on organization of storage and other decision-making functions, have had a significantly consistent tendency to fall into disuse. At the present state of knowledge on the effective use of large-scale general-purpose machines, it does not seem reasonable to attempt to formulate and develop into an automatic programming scheme a general theory of process-organization. The usual consequence of such an attempt is the crushing comment, "Yes, you have a clever system there, but unfortunately it won't fit the program I am working on now."

The *705* Autocoder is intended primarily to facilitate the coding of a particular machine, though it surely should be extensible to others of the same predominant general characteristic, that is, machines which are character-addressable rather than word-addressable. The Autocoder is intended as a tool to aid the human coder by assisting him in the ways in which an automatic coding scheme functions best, while leaving him on his own otherwise. Its principal features are these:

1. It allows the coder to define his constants and working data in terms of their familiar English-language names and their field lengths. He may address any item with which he is dealing by name and he may also refer to combinations of things, such as all the fields within a record under a single tag.

2. It extends the built-in *705* command code through the use of "macro-instructions," which are sequences of *705* instructions which experience indicates have a high-repetition rate.

3. It permits the incorporation of subroutines, which in turn may call in other subroutines, and so on. Since subroutines are written in Autocoder language it follows that successful programming becomes potential library material.

4. Since the Autocoder does not prescribe the organization of high-speed storage the programmer retains the freedom to establish any layout of instructions, constants, working and input-output space he desires.

Input to the Autocoder will be from punch cards generally, although tape input will be an alternative selection. Each input item, whether an instruction or the specification of a constant or datum field, will occupy a full card. This permits additional space for the coder's comment to be included with the required information. The Autocoder itself and its library of subroutines will be provided from a single tape.

The Autocoder assigns memory locations in order subject to the modification imposed by items in the input program which call for overlapping or skipping of selected areas of memory. In its processing of each input item it causes about 33 characters to be examined on a virtually character-for-character basis. This type of operation is feasible only because a character-addressable machine is available, and one which is quite fast in performing single-character operations.

The output from the Autocoder is a program in *705* actual, prepared in a form known as a load deck, that is, arranged in a form which permits rapid loading into the *705* memory, or onto drum sections or tape at the time it is desired to run the finished program. The Autocoder will produce such a load deck on cards or on tape alternatively. In addition there is produced a printed listing which displays in parallel the input program and the final result with actual memory locations.

The Autocoder requires three basic kinds of information to specify a program. These are:

1. *Instructions*. Commands to the *705* to perform certain logical operations on data or other instructions. These may be single *705*-type operations, or macro-instructions defining fixed sequences of *705* operations, or requests for closed-end subroutines.

2. *Constant definitions*. A listing of the required parameters, tables, and other constant information.

3. *Record definitions*. Detailed description and layout of input and output areas, and internally used storage areas.

A tag, consisting of a name or other descriptive designation, containing up to ten alphabetic or numeric characters is applied to any instruction, constant, or record field referred to by the program. Instructions which are not modified or transferred to need not be tagged. In addition to tagging individual constant or record fields, suitable combinations of them may be tagged also, as, for example, a PAY RECORD consisting of such fields as EMPLOYEE, SERIAL, GROSS PAY, and so on. Thus, it is possible to address whole records at a time with operations like READ, WRITE, or TRANSMIT (which is a unique *705* instruction for performing memory to memory transfers of any number of characters). A feature of the Autocoder is that it applies some logic to the operation itself to distinguish between operations involving fields within records, such as ADD and operations involving whole records, such as READ. The *705* requires that addresses be selected differently in these instances and the Autocoder adjusts accordingly.

Operations may have operands which are blank, to be supplied later by the program; operands which are actual, should it be desired to address specific memory locations; operands which are descriptive, as in addressing fields by their assigned tags; or operands which are literal. The use of literal operands is a novel feature of the Autocoder.

Literal addressing derives its name from the use of addresses which are the literal equivalents of the contents of the field being addressed. A literal address identifies a field in memory storage by specifying its contents. Other methods of addressing are "actual," "relative" and "symbolic," or "descriptive." In each of these systems a quantity stored in memory is addressed through some scheme of locating it, either relative to the entire memory, as in actual, or relative to some partition of memory, as in relative itself, or else by some unique label, as in symbolic. For example, the quantity +314159 (plus pi) may be addressed at its actual memory location, a relative location, or even at location PI as in SUBTRACT PI. However, the literal program employs SUBTRACT (+314159). The address is literally descriptive of the operand.

Whenever a programmer uses a scheme of addressing which is not literal he is saying, in effect: Operate on the quantity at the location designated by ————. In order to establish the proper location the programmer must maintain a record of location assignments which he consults whenever he refers to a constant or data field. The advantage of relative or symbolic addressing over actual is in the ease with which addresses may be assigned. However, the use of literal addressing obviates all look-up of address assignments. Thus computer coding is

brought a step nearer the spoken language which associates an action with its object.

By enclosing the literal quantity in parenthesis the coder tells the Autocoder that the address is literal. The Autocoder searches its list of already-stored literals, and unless the present one is a repeat assigns a new location for it. The number of characters within the parenthesis determines the length of the field established. Because the Autocoder does not allow the same literal to be repeated in memory it is possible to designate an area of working storage as well as a constant by this means. For, repeated references to any configuration of characters will result in the same actual memory location being addressed in the final program. Hence a variable may be stored and reused by addressing it in consistent fashion.

Areas of constants and records are described to the Autocoder by writing DEFINE CONSTANTS or DEFINE RECORD, using the proper codes, as operations, and then following with the fields comprising the area. Since the DEFINE line may itself bear a tag it becomes possible to refer to the whole as well as the parts. In the numeric column on the coding sheet is written the length of each field defined. On the DEFINE RECORD line itself may be written also the tag of some record previously defined. This instructs the Autocoder to overlap the present area on the one defined earlier. For example, on a single tape there may be two record types with different field arrangements. Both types will, of course, have been read into the same input area, but once having been distinguished, will require that their fields be addressed differently by the program. The ability to specify multiple overlays on the same actual memory area is the answer to this problem.

There is an advantage resulting from the descriptive tagging of record fields and from the fact that subroutines are written and stored in the same coding conventions as other programming. The expected situation in a given commercial data-processing installation is that there will exist a number of records which are common to many computer programs. For example, employee clock cards may be used by both the payroll process and some cost accounting procedures. Now, once the records are defined, the sequence of fields and their lengths established, it follows that each program and each subroutine within program may refer to the records by name. Hence it is possible for any number of coders to prepare library routines which refer directly to the common information of the installation. The actual filling in of the coding sheet and the subsequent key-punching has been made as minimal and as loose as possible. Nonsignificant zeros need seldom be written. Alphabetic fields are always written left-justified while columnar numeric fields are right-justified. This conforms to normal usage and should speed up training in the system while reducing clerical errors.

Although the *705* is itself rather intricately coded, requiring the specification of one of 40,000 characters of memory and a companion selection of one of 16 possible storage units in just four alphanumeric characters, the user of the Autocoder need not be concerned with the actual *705* addressing system. In fact, whenever he does refer directly to an actual *705* location he writes a 7-digit multiaddress which the Autocoder suitably encodes into the 4-character quantity recognized by the *705*. A projected system of macroinstructions should ease the task of address modification during the running of the program by providing the coder with an all-numeric, decimal logic with which to operate on an actual *705* address together with its binary superstructure.

The macroinstructions provide an exciting field for exploration. In a sense they allow the programmer to act as a creator of machine logic. The macroinstruction is like an open-ended subroutine in that it calls forth from a library of macrofunctions a sequence of *705* instructions which, under suitable modification, are inserted in its stead. However, the uses to which macro-instructions are being put and the concept of them seems quite new. They are regarded not so much as functions peculiar to specific classes of applications such as square root routines in mathematical applications, but rather as extensions of the built-in logic of the machine itself. For example, a common occurrence in running a machine is to type a message to the console operator. Ordinarily this requires the execution of a SELECT instruction to specify the output device, in this case the console typewriter, followed by a WRITE instruction, addressing some location at which a message has been stored. Instead of using these several instructions, the macroinstruction TYPE whose address is literally the message itself is introduced. The job is done in one line of coding instead of three, in this case. Even more powerful macroinstructions effecting input-output operations introduce tests for end-of-file, and error conditions. Should the latter occur, error-correcting routines are called automatically.

The two major categories of macroinstructions mentioned here so far are those dealing with address modification and input-output operations. In addition ways of causing the *705* to simulate other logics are being considered, such as floating decimal or fixed decimal with different scaling conventions than that built into the machine.

Also there is interest in the creation of additional commands which experience may indicate facilitate coding, such as a TRANSFER LOW, for example. Incidentally, this work should prove beneficial to the engineers who contemplate changes and new designs. The compound operations simulated most often are the ones the engineers would be encouraged to build into future equipment.

The *705* Autocoder is intended to be an aid to coding. It does not attempt to process more than an entry at a time, or to do more than the coder asks it to at any stage. The output program is under coder control throughout. Yet it is felt that this is a considerable asset, and it is felt that there are implications inherent in the Autocoder which have not yet been explored. An intriguing one is the possibility of direct and immediate transcription from flow diagram to computer input.

For example, in a process in payroll, on the flow diagram a box is found which directs the coder to SUBTRACT UNION DUES and the next to STORE GROSS PAY. But these are statements already in the language acceptable to the Autocoder. No further encoding is required. There may be subsequent boxes: WRITE CHECK RECORD and PRINT PAYROLL REGISTER LINE, all Autocoder language. The future possibilities for a thoroughly mechanical coding system are very favorable with a system like this for a starter.

However, it should not be inferred that a system as elementary as the *705* Autocoder is considered an end-all to machine usage problems. The preponderance of difficulty still resides in the systems-understanding and program-layout areas, especially for the commercial data-processing user. The technology and ingenuity in designing and running (including coding) of machines seems today well in advance of the understanding of what we are attempting to process with the machine. Automatic coding, of the kind exemplified by the *705* Autocoder, is a limited approach to a minor segment of a major problem.

Program Interrupt on the Univac Scientific Computer

JULES MERSEL

IN the use of high-speed digital computers the general situation has been to have the program or the programmer at the console in sole control of what the machine is doing. In recent years, however, a situation has often occurred where a program, so to speak, has only been lent the machine until a higher priority problem is ready to go on the machine. At these times the machine was stopped, the low priority problem was taken off the machine, and the high priority problem was allowed to run. Though such a situation is preferable to having a computer idle while it awaits the high priority problem, the time lost in getting the old problem off and the new problem on has been an unfortunate loss.

At the National Advisory Committee for Aeronautics' (NACA) Univac Scientific installation at Cleveland, Richard Turner foresaw that the foregoing situation would be the usual thing rather than merely an occasional nuisance. The way Mr. Turner intended to run his machine, and in fact does run it, the wind tunnel has first claim on the services of the Univac Scientific. It is important to NACA's operation that information from the wind tunnel reach the computer as soon as possible, that it be processed as soon as it reaches the computer, and that the results are returned to the wind tunnel immediately after processing.

The fast handling of wind tunnel information was achieved by running input-output lines directly from the wind tunnel to the Univac Scientific. Inasmuch as the Univac Scientific has the capability of having any kind of input-output, this was not a difficult task. The problem of quick access to the machine was solved by Mr. Turner's suggestion of the "interrupt" feature.

The interrupt feature has the following characteristics. When an interrupt signal is sent to the Univac Scientific, the computer, immediately after finishing its present instruction, notes the location of which instruction it would normally do next, jumps to a fixed address which will allow the value of this location to be stored and from which the computer will find the information as to what it is to do because of the computer having been interrupted.

As an example, in the wind tunnel operation, when the wind tunnel is ready to use the computer, an interrupt signal is sent from the wind tunnel. At the completion of the current instruction, the Univac Scientific notes where it is in the current problem, stores the current problem away, reads in the program for the wind tunnel problem, and starts that program. The wind tunnel problem then reads in its own input from the wind tunnel, executes the needed computation, and returns the answers to the wind tunnel. As soon as the answers are returned it reads the previous problem back into the rapid access storage and continues it from the point where it had been interrupted. A delay of microseconds has been incurred in getting ready for the switchover rather than the much lengthier delay which might be expected without the use of a program interrupt.

This application of a program interrupt inspired much thought as to other applications on which the program interrupt might profitably be used. The wind tunnel application is an input-output application with a device that is not normally considered part of a computer. Thought was naturally given to whether the program interrupt might be profitably used with ordinary input-output equipment. Such pieces of peripheral gear as punched card input and output equipment turned out almost to demand the aid of the program interrupt.

In handling punched cards the normal procedure is that, for 12 very short intervals during the whole card cycle, information must be either read from the rows of the card or punched onto these rows or both. Since on the Univac Scientific less than 1 per cent of the time of the card cycle is required for the reading or writing of rows, the programmer usually attempts to perform computation during the remaining time. This, unfortunately, requires that he carefully segment his computation into 12 groups none of which is time-wise long enough to interfere with the processing of row information. This is an onerous task and usually results in either safe but inefficiently short segments of computation or segments that cause machine errors due to being too long.

The card input-output device on the Univac Scientific was modified so that it would send an interrupt signal when it was ready either to release or receive row information. The need for the careful segmentation of the computation disappears. When the interrupt signal is received, the computation is stopped, the row information is either written, read, or both, and then the computation is continued from the point of interruption. The elimination of the need for careful timing of computation usually allows much more computation to be performed during the card cycle.

Probably the most interesting uses of the "program interrupt" center around the task of using two computers on the same problem.

Some Univac Scientific users intend to use their machine in conjunction with analogue computers. In this case the analogue computer interrupts the digital computer whenever it is ready to transfer information. The Univac Scientific then processes the analogue data, returns it if needed to the analogue computer, and then awaits new data from the analogue computer. If much delay is expected in receiving information from the analogue computer, computation on other problems may take place during the wait. This, of course, greatly increases the efficiency of use of the computer.

One user is planning to use two Univac Scientifics in tandem. His is a real time problem. One computer smoothes the raw data that comes in from the outside; the other computer processes the smoothed data. In such a case whenever the second computer is ready to receive more data it interrupts the first computer and through the Univac Scientific's versatile input-output system the smoothed data are transferred from the first machine to the second machine. This allows the maximum amount of smoothing to take place within the time limits of the problem. Many other applications can be visualized where in real time problems it is necessary to use two computers in tandem since for these applications no one computer is fast enough to do the job by itself. Since real time problems at such installations would occupy only a small part of the computing day, the two computers are freed to be used separately during the rest of the day.

The last application occurs when, because of the form of graphs displayed on

JULES MERSEL is with The Remington Rand Corporation, St. Paul, Minn.