Print I—A Proposed System for the IBM Type 705

R. W. BEMER

Purpose of the Print I System

THE PRINT I (PRe-edited INTerpretive) system has been designed to meet the engineering and scientific computing needs of those Type *705* installations where such work is a secondary computing requirement. It is specifically tailored for this purpose and in general has no similarities to Type *705* systems designed for business and commercial applications. Although provision is made to move freely from abstraction to *705* commands and back, it is not recommended that PRINT I be tied to any business system for the reason that the restrictions of one type of usage will too often hamper the other.

The basic consideration in the planning of this system has been ease of learning and continued operation by personnel with either very modest programming experience or none at all. Pseudo-instructions are simple and straightforward; basic principles may be assimilated quickly. There are no restrictions in the use or combination of pseudo-instructions, although minor increases in operating speeds have been introduced to effect this. Basic logical errors in the written program will be detected automatically and detailed analysis of such errors will be typed out to the operator.

Choice of System Characteristics

Components

Since some Type *705* installations have ordered configurations which do not include a magnetic drum, the only components specified for this system are magnetic core memory and sufficient magnetic tape units to handle expected problem size.

Abstraction Type

The first decision to be made was between the compiling and interpretive modes. The fastest method of computer operation is with a compilation of basic machine commands operating in linear progression without modification. The first drawback to this method is that there are no computers in production with the nearly infinite memory required for this long thin line of instructions. The second is that the time advantage gained is not worth enough to compensate for cost of additional memory if it were available. A more practical variation of this is the type of compiler which still forms machine commands but uses modifying commands to reduce the volume of instructions, just as a conscientious programmer would. The serious drawback to this method is that it requires that extreme complexity be built into the compiler in order for it to function as efficiently as the clever programmer. In addition, the program produced is often much larger than that produced by a compact interpretive system.

An interpretive mode offers quick construction and minimum storage, obtained by the complicated weaving and interlocking that the coder may accomplish. Its drawback, an extremely serious one, is that the command is normally re-interpreted every time it comes up for execution in actual operation. Thus the variable "fetch" and "operate" commands are fabricated a multiplicity of times, whereas a compiler fabricates them only once.

This has led to rather determined stands by the opposing compiling and interpreting adherents. Fortunately there is a compromise available, as demonstrated in the PRINT I system. A pre-editing routine performs a compiling and assembly function to make the commands numerically palatable to the executive routine which is in the interpretive mode. In addition, a repeat command is furnished in the list of pseudo-instructions. This enables certain commands to be executed n times in succession, once with interpretation and command fabrication, and $n-1$ times in the exact fashion that an ultra-efficient compiler would generate the program. Multiaddress commands are used, for although single-address commands are more efficient when the desired characteristics are inherent in the machine language command, multiaddress commands are more efficient in an interpretive abstraction, just as buying more groceries at a time minimizes the cost of trips to the store.

Arithmetic and Format

Floating decimal arithmetic was chosen as most generally acceptable to all scientific users. Since the specification of a mantissa length must, to our way of thinking, specify a corresponding set of subroutines to that accuracy, PRINT I will be packaged initially in 8- and 10-digit versions. A 20-digit version is to be produced after completion of the 8- and 10-digit systems. If enough demand should exist, a 5-digit version may well be produced. Each system will be complete in itself for all operation. There will be complete freedom of interchange between the floating point abstraction and *705* language.

For faster internal operation and convenience, floating point numbers are stored internally as $\pm\ \pm$ *xxx. . . . .xxPP*. The *x*'s represent a mantissa which is a proper decimal fraction with a non-zero leading digit. *PP* is the power of 10 multiplying this fractional number. Externally, the format is $\pm\ \pm$ $\pm PP \pm xxx\ldots$, so that trailing zeros need not be written by the programmer; the pre-edit routine will take care of this automatically.

As far as possible, all arithmetic operations and subroutines will be performed with rounding. A legitimate zero in this system has both power and mantissa equal to zero. To facilitate and accelerate operation with zero operands, the mantissas will be tested for zero in all arithmetic operations. Messages are provided to be typed out in case of error during operation, such as:

Division by zero.
Square root of a negative number.
Power overflow (exceeding +99).
Logarithm of zero or a negative number.
Sine or cosine of angle greater than a prescribed limit.

This list of error messages will be completed as the various elements of the system are finished.

All addressing of operands and locations of pseudo-instructions for this system are in regional form, consisting of one leading alphabetic character and, normally, three numeric characters. Inserts may be made by an additional numeric character on the right, as *G*125 and *G*1251.

Tracing and Diagnostic Routine

There will be a single type of diagnostic routine associated with this system, since the programmer will want to see

R. W. Bemer is with the International Business Machines Corporation, New York, N Y.

results corresponding to his pseudo-instructions, and since compound indexing introduces time-wise complications, a trace routine of an insert transfer to trace control type will be used. This routine will allow high-speed operation to various points of interest, at which time tracing of a specified nature will occur. This may mean tracing of every pseudo-instruction, stores only, transfers only, or any combination of these. This will be under the control of alteration switches. When under trace control, tracing will proceed indiscriminately through both PRINT I and *705* commands.

Interpretive Section

PRINT I will always be in core memory during operation of a program prepared for this system. Work done so far indicates that the arithmetic operations and standard subroutines should be contained within 4,000 character positions. Provision is made for a floating subroutine to be called from tape storage as required. This is described in detail in the section on the pre-edit routine.

The interpretive routine does not use check indicators in any way as decision elements. They are reserved for stops while operating in *705* language, and switches may therefore be set to automatic stop during the operation of PRINT I. Any entry to PRINT I sets up the auxiliary storage units as required for its operation; therefore, all auxiliary storage units are available while operating in *705* language.

Indexing

A system of 4-dimensional indexing is simulated within PRINT I. This indexing is incremental and transfer is dependent upon the contents of the registers exceeding a specified limit. There are three index registers, referred to as *R*1, *R*2, and *R*3. They are four digits in length and the contents may be either positive or negative. Since these contents are used only for address modification, they are unsigned, and negative quantities are stored as 40,000 complements. Any regional address may be augmented by the contents of any of these registers or the arithmetic sum of any two or all three. This alteration takes place in a fixed memory position before fabricating the necessary machine language commands from the address portions of the pseudo-instructions. The original command in operating sequence is never altered. Loops formed by transfer on index commands will therefore require reindexing from the original addresses. Proper use of indexing features will greatly lower the number of PRINT commands in any program. It is for this reason that the interpretive routine of up to 4,000 characters can be tolerated as a permanent part of core memory.

System Entry

PRINT I will be loaded into core memory from a deck of cards furnished for a specific mantissa length. Tape unit *1* is normally assigned to contain, for purposes of re-entry, the PRINT system after initial loading. It will also contain the pre-edit routine, nonstandard subroutines and both the original and edited progams. On entry of the system from either cards or tape, a general check will be performed to indicate proper loading. If errors occur, typed-out alarms will indicate their nature; a second loading may in some instances be entirely satisfactory.

Input and Output

Two types of input are required, one for instructions to the pre-edit routine and the other for data necessary for execution. Data input routines of three types may be provided, as follows:

Serial loading, based on a single specified address per card.
Random loading, address and contents specified for each word.
Special 3- or 4-digit volume data loading, a single floating power being assigned to all input and scaled in the program.

Output to printer will be by type-wheel image in memory, consisting of several words appropriately spaced for printing. These positions will be loaded by pseudo-store-for-print commands specifying data position number. Printing will be of the write-erase 00 type for flexible format. Spacing will be under program control with write-erase 01 specifying a group mark in the highest memory position.

Punching, when necessary, will be accomplished by a similar method, utilizing a card image position in memory.

Pre-edit Routine

The programmer will normally code mnemonic commands which specify, in a variable field controlled by commas, the regional addresses and indexing. A typical command is:

SUB F123, 1, F286, 2, F174, 1+3

These commands and associated comments are punched one per card for pre-edit and assembly. Card columns are punched consecutively, the first blank column indicating the end of the command and the start of the comment field. For each mnemonic command, the pre-edit routine produces a corresponding numeric command especially tailored for the fabrication of machine commands from its components. This numeric command will be of varying length according to the operation specified. Matched sets of mnemonic and numeric commands, together with the comments, may be printed for inspection at pre-edit time at the option of the operator. If the program is known to be correct, the edited form may also be punched out on cards for later re-entry in more condensed form, also avoiding repetition of the pre-edit process. Both of these options are under alteration switch control. For convenience in correcting programs, both the original mnemonic program and the edited routine will be stored on the library tape. Temporary patching of the program, by transfer-out and return, may be made by reading in command cards to replace those in error (this group may contain inserts). After the program is running correctly, these patch cards may be inserted in their proper sequence in the program and the entire program re-edited properly. Tracing will be printed out in conformity with the pseudo-instructions as far as possible, with respect to the contents of the various addresses and index registers.

A regional address notation was chosen to insure early completion of this system. Symbolic addresses of the type where the address is either descriptive of the contents or literally the contents themselves would greatly complicate the pre-edit routine. Complete freedom of interchange between *705* commands and PRINT I commands is automatically assured by the pre-edit routine. Upon discovery of variance from, or return to, the 5-digit *705* command, the pre-edit inserts links and transfers as required. For this reason, any address in a non-alphabetic region is a fixed *705* character address. The *Z*-region is reserved for PRINT I.

Two types of regional addresses are recognized by the pre-edit routine. The first is for data, where the address represents a fixed number of characters according to the mantissa length of the system used. Data should normally be allocated by the programmer to the highest portion of memory. This is done by specifying to the pre-edit routine that a certain region starts at a specific character address, the extent and positioning of the region being dependent on (number of addresses in region) times (word length). The basic address of a data word is that of its highest memory position. The pre-

edit allots memory in $(m+2)$ modules, where m is the mantissa length, and computes the basic address from the numeric portion of the address.

The second type of regional address is for the pseudo-instructions, which are of various lengths. They are normally allocated to the portion of memory just above the PRINT I system, and are normally obeyed in ascending order. The basic address of an instruction is that of its lowest memory position and equals (the basic address of the previous instruction)+(length of previous instruction). In this case, the pre-edit must construct a table to correlate the regional address of each instruction with its basic address. If the program should start to infringe upon memory allocated to data, the pre-edit types an alarm indicating the overlap position. Automatic treatment of this situation is not provided for; responsibility for the avoidance of or correction of such a condition is assigned to the programmer.

The programmer is not required to write redundant information in the variable field of the pseudo-instruction. The pre-edit automatically inserts the proper information as described under the command summary.

Certain alarms are set up to be typed out during pre-edit if the programmer has ignored the very few restrictions inherent in the PRINT system. A partial list of alarms contains:

Program running into data storage.
Calling for a pseudo-instruction as data, when not in PRINT, with the exception of the commands replacing first address.
Nonrepeatable command following a repeat command.
Nonalphabetic address ending other than 4 or 9.
Minus index limit on a clear index and set limit command, or converted limit greater than (memory capacity—10,000).

When a floating subroutine is coded, the pre-edit knows that such a mnemonic symbol has no assigned operation code in the table. It is assigned the floating subroutine operation code and the pre-edit automatically sets up *705* instructions to bring the proper subroutine (if it exists on the library tape) into the floating position in PRINT I. Such a call-out is set up on change of requirement only. Depending on detailed study of the various factors in frequency-time-storage balance, it may be desirable to set up two such floating subroutine positions. Another possibility is for the programmer to specify to the pre-edit routine the amount of memory he is willing to allocate to floating subroutines. Replacement would then be set up only if the desired subroutine exceeds in size the amount of available memory left.

Operation Summary

General

Operation codes are limited to 40 in number. This does not necessarily limit the number of operations to 40, for a single operation code may be tagged in the body of the command. Twenty plus codes, from 04(5)99 are allotted for nonindexable operations. Twenty minus codes, from $-99(5)-04$ are allotted for indexable operations. Proposed codes and corresponding mnemonic symbols are:

Nonindexable			Indexable		
04	CS1	54	$\overline{04}$	TMT, TAB	$\overline{54}$ SQR
09	CS2	59	$\overline{09}$	ADD	$\overline{59}$ SIN
14	CS3		$\overline{14}$	SUB	$\overline{64}$ COS
19	TRZ, TNZ, TR+, TR−		$\overline{19}$	MPY	$\overline{69}$ ART
24	TRU		$\overline{24}$	−MY	$\overline{74}$ LGD, LGE
29	RPT	79	$\overline{29}$	DIV	$\overline{79}$ EXD, EXE
34	RWR	84	$\overline{34}$	−DV	$\overline{84}$ TRC
39	TX1, TN1	89	$\overline{39}$	MAD	$\overline{89}$ TRE
44	TX2, TN2	94	$\overline{44}$	PMA	$\overline{94}$
49	TX3, TN3	99	$\overline{49}$	FSR	$\overline{99}$

Other operations (such as ELO, NOR, various store-for-printing operations, and placing the TRx tally in an index register) are not fixed at this time and are not shown in the table. If desired, the user may insert his own subroutines by placing the proper entry in the operation switch position; this may involve replacement of some of the standard subroutines.

This manual will contain an appendix with enough detailed information to enable the user to make these and other alterations with relative ease.

Interpretation and indexing times vary with the indexing requirements as shown in table (time given in milliseconds):

Type	No Indexing	1 field	2 fields	3 fields
Test first for any	0 816	1 326	1 581	1.836
Test each in order	0.952	1.207	1.462	1.717

One of these two types will be chosen on the basis of lesser elapsed time in actual operations. Nonindexable commands require 0.697 millisecond (ms) for interpretation.

α, β, and γ may be any regional address, including α, β, and γ or the pseudo-accumulator PACC, which is a field in memory reserved for that function. PACC is not indexable, and a zero indicator is automatically inserted for it by the pre-edit routine. Indexable commands are usually written with the index register indications for each address following the address itself, separated by commas, as:

SUB G258, 2, G184, 2+3, G702, 1+3

If the result address is omitted in writing the pseudocommand, the pre-edit routine assumes that it is to be PACC. If no index information is written, indexing is denied by setting the tag to zero.

Indexable Operations

Class I-1, Arithmetic Operations. There are eight operations in this class. The operations are:

Written	Signifying
ADD α, $R\alpha$, β, $R\beta$, γ, $R\gamma$	Place $(\alpha)+(\beta)$ in γ, PACC
SUB	Place $(\alpha)-(\beta)$ in γ, PACC
MPY	Place $(\alpha)(\beta)$ in γ, PACC
−MY	Place $-(\alpha)(\beta)$ in γ, PACC
DIV	Place $(\alpha)\div(\beta)$ in γ, PACC
−DV	Place $-(\alpha)\div(\beta)$ in γ, PACC
MAD	Place $(\alpha)(\beta)+(\text{PACC})$ in γ, PACC
PMA	Place $(\alpha)(\text{PACC})+(\beta)$ in γ, PACC

Two mnemonic symbols may require explanation; MAD stands for multiply-add and PMA stands for polynomial multiply-add, so-called because of its suitability to polynomial evaluation with α as the address of the argument. These two operations are also exceptions in that when modified by a RPT command the result is not stored in γ until the last repetition is complete.

Class I-2, Subroutine Operations. There are nine operations in this class. One of these is a floating subroutine which is at any time the function assigned to it by the pre-edit routine. At present, all

transcendental functions refer to radian arguments. The operations are:

Written		Signifying	
SQR	$\alpha, R\alpha, \beta, R\beta$	Place $\sqrt{(\alpha)}$	in β
SIN		Place sine (α)	in β
COS		Place cosine (α)	in β
ART		Place arctangent (α)	in β
LGD		Place $\log_{10}(\alpha)$	in β
LGE		Place natural log (α)	in β
EXD		Place $10^{(\alpha)}$	in β
EXE		Place $e^{(\alpha)}$	in β
FSR	↓	Place required function of (α)	in β

Class I-3, Transmittal Operations. There are two operations in this class. Both place the contents of α in address β. When used with an RPT command, they will effect block transfer in a direct or inverse manner, or open up a series so that interpolated values may be interspersed. For the TAB (transmit absolute) operation, the number is guaranteed to be positive in the β position, remaining in the original form in α. Unless PACC is specified as the β address, it will be unaffected by either of these operations, which are:

Written		Signifying
TMT	$\alpha, R\alpha, \beta, R\beta$	Place (α) in address β
TAB	$\alpha, R\alpha, \beta\ R\beta$	Place the absolute value of (α) in address β

Class I-4, Comparison Transfer Operations. There are two operations in this class. They are:

Written		Signifying
TRC	$\alpha, \beta, R\beta, \gamma, R\gamma$	Transfer to command in α if (β) are greater than or equal to (γ)
TRE	$\alpha, \beta, R\beta, \gamma, R\gamma$	Transfer to command in α if $(\beta)=(\gamma)$

When either of these commands is modified by an RPT, a special addressable register (4-digit) is reset to zero and a 1 is tallied in it every time the transfer condition is not met. γ is normally not indexed by the RPT and is the item to which successive β's are compared. This allows for block table look-up and later modification to a finer interval of search. This may be done by *705* language modification of both the RPT and the TRx command to alter both the starting address and the search interval. A preferable method is to use a new pair of commands with the RPT already set to the finer interval and the TRx command indexed by a function of the tally. For such TLU operations the n of the RPT command is normally set to 99.

Class I-5, Floating Point Operations. There are two operations in this class. They will be used to convert fixed point numbers to floating point form and vice versa. Exact characteristics have not been specified yet. The operations are:

Written		Signifying
FLO	$\alpha, R\alpha, \beta, R\beta$	Convert the fixed point number in α to floating point form and place in β
NOR	$\alpha, R\alpha, \beta, R\beta$	Convert the floating point number in α to fixed point form and place in β

Nonindexable Operations

Class N-1, Transfer Operations. There are five operations in this class. Although they in some respects duplicate *705* commands, they are nevertheless included in PRINT for convenience. They operate in less time than it would take to leave to *705* language, test, transfer, and re-enter PRINT. There is also a distinct saving in memory positions. The operations are:

Written		Signifying
TRZ	α, β	Transfer to command in α if $(\beta)=0$
TNZ	α, β	Transfer to command in α if $(\beta)\neq 0$
TR+	α, β	Transfer to command in α if (β) are +
TR−	α, β	Transfer to command in α if (β) are −
TRU	α	Transfer to command in α unconditionally

Class N-2, Repeat Operations. There are two operations in this class. The operations are:

Written		Signifying
RPT	n, i, j, k	Perform the next command n times, using as operands the contents of $\alpha+(n-1)i$, $\beta+(n-1)j$ and $\gamma+(n-1)k$, as required.
RWR	n, i, j, k	Same as above, except reset PACC (pseudoaccumulator) to zero before proceeding.

These operations apply only to the next operation. By letting the interpretive routine know in advance, and indexing by given numbers rather than the contents of an address, the repetition operates at maximum speed, requiring neither interpretation or command fabrication. Indexing by repeat commands is secondary and subordinate to indexing by index registers, and the simultaneous use of both is possible. Operands are then the contents of:

α indexed $+(n-1)i$, etc.

n is a one or two digit number, always plus but carried negatively in memory. i, j, and k may be 1- or 2-digit numbers, both plus and minus. When minus, they are carried in memory as unsigned 40,000 complements.

All repeatable (indexable) instructions interrogate the n position, which serves as a tally, before storing a result. If this is nonzero the tally is reduced by one and the operation is automatically repeated with further indexing by i, j, and k. If this is zero, it signifies either that the command was not intended to be repeated or that the command has been performed for the nth time. In either case, the program proceeds to the next command in sequence. In the case of the MAD and PMA commands, the result is not stored in γ until the tally is zero. Intermediate results are stored in PACC.

Class N-3, Set Index Operations. There are three operations in this class, one for each of the index registers. Each resets the contents of the specified index register to zero and stores the quantity j from the command as a limit tally. j is always a positive quantity and when converted (by multiplying by data word length) must be less than or equal to (the memory capacity of the *705*) $-(10{,}000)$. The operations are:

Written	Signifying
CS1	Reset R1 to zero and set limit tally to j
CS2	Reset R2 to zero and set limit tally to j
CS3	Reset R3 to zero and set limit tally to j

Class N-4, Non-Test Transfer Operations. There are three operations in this class, one for each index register. Each augments the contents of the specified index register and the corresponding limit tally by the quantity k, which may be both positive or negative, and transfers to the instruction in α. To increase operating speed, all possible combinations of sums of index registers are also carried along and the appropriate combinations are also augmented by the quantity k. This permits single-indexing for any address. When converted, k is stored in either true or 40,000 complement form for unsigned add-to-memory. The operations are:

Written		Signifying
TN1	α, k	Augment all registers using R1 by k and transfer to the command in α
TN2	α, k	Same as above, except refer to R2
TN3	α, k	Same as above, except refer to R3

There are two methods of setting an index register to an arbitrary value. One is to give a CS1 and then a TN1 with that arbitrary value in the k position. The other, used when the present contents of the register are known, is to give a TN1 with k as the difference between the known and desired contents. The trans-

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

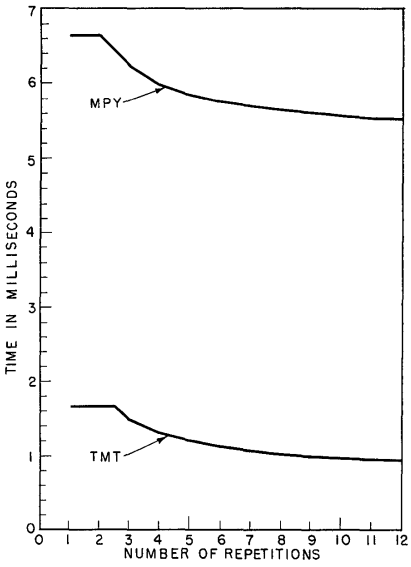

Fig. 1

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.