Kolsky

[redacted]

BULLETIN 3

00039

BINARY INPUT AND OUTPUT

for the

IBM

ELECTRONIC

DATA PROCESSING MACHINES

TYPE 701 AND ASSOCIATED EQUIPMENT

INTERNATIONAL BUSINESS MACHINES CORPORATION
APPLIED SCIENCE DEPARTMENT
New York, New York

The following information is
not final and is subject to mod-
ification. The information is
the best available at this time.
August, 1952

FOREWORD

The programs presented in this bulletin form a part of a comprehensive system of information handling for the IBM Electronic Data Processing Machines, and all programs shown have been used successfully on an installation of this equipment. However, the system is subject to modification whenever more efficient techniques are discovered, and the programs are presented primarily to illustrate typical techniques of operation in such a system. At present plans are being made to include a check sum, half-word count, and loading address in the nine row of each binary-punched card.

Treatment of Binary Information

It is frequently convenient to read or write information in binary form using the IBM Type 701 and associated equipment. Since information can be stored more compactly in binary than in decimal form, the quantity of storage required for a given block of information is reduced. Correspondingly, reading and writing speeds are increased by the use of the binary number system. For example, information may be entered into the IBM 701 at the rate of 1050 ten-decimal-digit words per minute by means of the card reader using decimally-punched cards. Each card contains seven ten-decimal-digit numbers and signs. However, since a punched card may contain twenty-four 35-bit words and signs, information may be entered from binary-punched cards at the rate of 3600 words per minute.

Binary information must be entered into the IBM 701 when no suitable loading program is available in electrostatic storage for the entry of decimal numbers. Such a situation is always encountered at the start of a day's operation.

This report presents four programs to illustrate techniques that have been found useful in the treatment of binary information. Self-Loading 5 (SL 05) is a typical self-loading program which includes a sequence for the loading of additional binary instructions or data from punched cards. Binary Punching 5 (BP 05) and Binary Punching 6 (BP 06) may be used to produce cards punched with the contents of electrostatic storage in such form that the cards may be read by SL 05. Read-Write 2 (RW 02) is a general read-write program suitable for reading or writing cards, tapes, or drums with binary information.

Self-Loading 5 (SL 05)

The principle of self-loading has been discussed in the Preliminary Report on the IBM Electronic Data Processing Machines (pp. 42-46). The programs shown in that report may be used to load electrostatic storage with information punched in cards in binary form. However, those programs require that every program to be loaded carry its own self-loading sequence and that all loading be done using the consecutively-numbered electrostatic storage locations immediately following the locations occupied by the self-loading sequence itself. It is evident that these restrictions need not apply in general, since a self-loading sequence of six instructions may be used to enter a more general loading program. SL 05 is an example of such a program.

Self-Loading 5 is contained in a single card. It includes the six instructions necessary to allow it to be entered by the operation of the LOAD button. It makes use of electrostatic storage locations 0000 through 0047. Self-Loading 5 may be used to enter binary information from any number of punched cards. This information may be stored in any desired consecutively-numbered electrostatic storage locations, except those already reserved for Self-Loading 5, and the reading and storing operations are completely checked by use of a check sum. The following quantities must be specified for each block of information to be loaded:

Card Check Sum: This is calculated by summing the absolute values of all half words in the block, except the two half words forming the card check sum itself, plus 2^{17} times the number of negative half words, doubling this sum, and attaching a minus sign. This card check sum requires a binary full word and is punched in the left half of the nine row of the first card of the group containing the block of information to be loaded.

Half-Word Count (V): This is the number of half words to be loaded into consecutively-numbered locations. This count does not include the two half words of the card check sum, the half-word count, or the initial loading address. This count requires a binary half word and is punched in the sign position and positions 1 - 17 of the right half of the nine row of the first card.

Initial Loading Address (R^1): This specifies the locations into which the following block of information is to be loaded. The V half words of the block occupy electrostatic storage locations $R^1 + 2$ through $R^1 + V + 1$. Locations R^1 and $R^1 + 1$ are reserved for the Storage Check Sum whose calculation is described in the discussion of the program's output. R^1 must be a positive even integer. It requires a binary half word and is punched in positions 18 - 35 of the right word of the nine row of the first card.

Note the following:

1) Both V and R^1 are included in the calculation of the check sum contained in the card.

2) Since SL 05 occupies electrostatic storage locations 0000 through 0047, R^1 must be greater than or equal to 0048.

3) If more than one block of information is to be loaded, then the following two requirements must be observed:

a) The V half words of each block must be loaded into consecutively-numbered electrostatic storage locations; and

b) Each block must start on a new card whose nine row contains the card check sum, the half word count, and the initial loading address for that block.

4) If V is odd, location $R^1 + V + 1$ is set to zero.

The output of the program consists of V + 2 half words for each block of information entered. These half words are stored in locations R^1 through $R^1 + V + 1$, the first two of these half word locations being reserved for a full-word storage check sum. This sum is calculated by summing the absolute values of the V remaining half words of the block together with 2^{17} times the number of negative half words, doubling this sum, and attaching a minus sign. It will be seen that the storage check sum minus 2V minus $2R^1$ equals the check sum read in from the first card of the group.

Operation of the Program:

The card containing SL 05 is put at the front of the file of cards containing one or more blocks of binary information to be loaded. This file of cards is placed in the card reader, the address entry keys are restored to zero, and the load button is depressed. The rest of the loading operation takes place automatically under calculator control.

The first six instructions of SL 05 accomplish the loading of the remainder of this program. Control is then transferred to location 0006 as a result of the end-of-record skip. Instructions 0006, 0007, and 0010 read and copy the nine row of the first card whose information is to be entered into electrostatic storage. This row contains the card check sum and the quantities V and R^1. These words are stored in locations -0002, +0004, and +0005 and replace four instructions of the loading sequence, which is no longer required. The card check sum, V, and R^1, are then used by instructions 0011 through 0023 to reset various address parts and to calculate the storage check sum which is stored in locations $-R^1$ and -0002.

The remaining V half words of the block are then read and stored by instructions 0024 through 0032. When this information has been entered, control is transferred to location 0033. Instruction 0033 is the first of the loop, 0033 through 0043, which verifies that the newly-entered information is stored correctly. This verification is performed by the addition of successive half words, whose signs are treated as described above, to the storage check sum stored in location 0002.

After the V half words have been added to the storage check sum, instruction 0044 tests this new sum for zero. If no error has been made, this test for zero is successful, and control is transferred to location 0006. The calculator is now ready to read a second block of information. This process continues as long as no error is made and cards remain to be read.

After the reading of the last group of cards whose binary contents are to be stored, the program verifies the correctness of the newly-entered information and again transfers control to location 0006. At this point, however, there remain no cards to be read. Thus the end-of-file skip causes control to be transferred to location 0009, which contains a STOP instruction. This signals the successful completion of the loading operation. Should an error be detected in the summing of a record, the calculator stops at instruction 0045.

Self-Loading 5, once loaded, may be used any desired number of times without reloading simply by placing the cards to be read in the card reader and starting the calculator at instruction 0006.

Binary Punching 5 (BP 05)

This program is designed to operate in conjunction with Self-Loading 5 (SL 05) for the purpose of preparing binary cards whose contents can later be reentered by SL 05. The information to be punched by a single execution of BP 05 must occupy consecutively-numbered half-word locations. This block of locations may start with any even location R, provided that R is greater than or equal to 0094, and may include V locations, where V is any even integer. The check sum required by SL 05 is computed and punched by BP 05. Thus alterations may be made on a block of information in electrostatic storage without regard for any changes that these alterations may make in a previously-calculated check sum, since a new check sum is calculated automatically prior to punching. A program may be entered without a check sum, as for instance from decimally-punched cards, and this program may then be punched in binary cards by BP 05 in form suitable for loading by SL 05.

BP 05 is entered by SL 05 into electrostatic storage locations 0050 through 0093. The half-word constants R, V, and R^1 are then placed in locations 0047, 0048, and 0049. These three constants specify the first location of the block to be written (R), the number of half words in this block (V), and the first location the block is to occupy when reloaded by SL 05 (R^1). The program is then called in by a transfer of control to location 0050. The execution of the program is fully automatic. The above operations need not be consecutive, but the presence of SL 05 is required in the execution of BP 05, and, unless the last previous execution of SL 05 was in the loading of BP 05, the integer 44 must be entered in half-word location 0004.

Operation of the Program:

The instructions stored in locations 0050 through 0059 make use of R, V, and R^1 to reset various address parts in preparation for the execution of the remainder of the program. These instructions also calculate $-2(V+R^1)$ and store this quantity in the location (-0002) to be occupied by the check sum.

Instructions 0060 through 0070 calculate the full-word check sum to be punched in the left half of the nine row of the first card produced. Since this sum is to be used by SL 05, a description of its calculation may be found in the discussion of that program.

Following the calculation of the check sum, the WRITE instruction is given for the first card. The first row punched is the nine row, containing the check sum, V, and R^1. The remaining rows of this card are then punched with the first 44 half words of the block to be written, provided that V is greater than or equal to 44. If V is less than 44, only the minimum number of COPY instructions necessary for the punching of V half words are executed. Instructions 0077 through 0085 perform this punching operation.

If V is greater than or equal to 44, control passes to location 0086 following the punching of the first card. A test is then made by instructions 0086, 0087, and 0088 to determine whether the entire block has been written. If more information remains to be written, the WRITE instruction in location 0089 is executed, and the punching of the next card is begun. Since tests are made both during the punching of a card and before the interpretation of a WRITE instruction to determine whether the punching process is complete, no COPY or WRITE instructions are given beyond those required to write exactly V + 4 half words.

When all information has been punched, control is transferred to instruction 0093 which stops the calculator. This signals the end of the operation. The address part of the STOP instruction is 0050, so that the check sum calculation and the punching operation may be performed for a second block of information by inserting new R, V, and R^1 into locations 0047, 0048, 0049 and pressing the START button. R must be reentered if these operations are to be repeated for the same block.

The operation of BP 05 may be checked by inspecting V and R^1, punched in the right half of the nine row of the first card, and by using SL 05 to load the set of cards just punched. The operation of SL 05 verifies the check sum.

Binary Punching 6 (BP 06)

This program is designed to punch cards with binary information in such form that this information may later be reentered by SL 05. The information to be punched by a single execution of BP 06 must occupy consecutively-numbered half-word locations, and the first two half words of this block, stored in locations R and R + 1, must contain a storage check sum as described in the discussion of SL 05 output. R may be any positive even integer such that the locations assigned to contain the block of half words to be punched do not conflict with those containing the instructions or associated constants of BP 06.

BP 06 is called in by a calling sequence of the following form:

α	R ADD	α	Basic linkage
$\alpha + 1$	TR	0048	
$\alpha + 2$	STOP	V	Half-word count
$\alpha + 3$	STOP	R	Unloading address
$\alpha + 4$	STOP	R^1	Reloading address

In this sequence three constants are specified: the number of half words (V) to be punched, not including the two half words of the check sum; the unloading address (R) which is used to calculate the location of the information upon which BP 06 is to operate; the reloading address (R^1) which determines into which locations the binary information is to be reloaded by some loading program such as SL 05. Following the execution of BP 06, control is transferred to location $\alpha + 5$ of the calling sequence.

The output of the program consists of a set of binary cards. The first card contains the card check sum, V, and R^1 in its nine row. The remaining 44 half-word positions of the first card, and all 48 half-word positions of succeeding cards, contain the V half words of the block of information whose punching was desired in the execution of BP 06. The V + 2 half words upon which BP 06 operates are unaltered by the execution of the program. Thus the program may be used any desired number of times for the punching of a given block of information. In its execution, BP 06 verifies that the information which is punched has been remembered correctly in electrostatic storage. This is done by adding each half word of the block to the storage check sum, in the manner described in the discussion of SL 05, and testing this reduced check sum for zero after all half words of the block have been added to it.

Operation of the Program

The instructions stored in locations 0048 through 0055 calculate and store addresses in the calling sequence, including the locations of V, R, and R^1. Instructions 0056 through 0059 store V and R^1 in locations 0122 and 0123. Instructions 0060 through 0068 make use of R to reset various address parts in preparation for the execution of the remainder of the program. Instructions 0069 through 0073 subtract $2(V + R^1)$ from the storage check sum, stored in location -R, thus obtaining the card check sum in the form used by SL 05. This newly-calculated sum is stored in location -0120.

The WRITE instruction stored in location 0074 selects the card punch. Following the execution of this instruction, the quantity (R + 46) is stored as the address part of instruction 0113 by the execution of instructions 0075 through 0077. Instruction 0113 is never executed, but is used to ensure that no more than twenty-four COPY instructions are executed for any card punched under the control of this program. The card check sum, V, and R^1 are punched in the nine row of the first card by the execution of the two COPY instructions in locations 0078 and 0079.

The program then begins the execution of the COPY sequence, instructions 0080 through 0088, which executes the COPY instructions for the remaining 22 half rows of the first card if V is equal to or greater than 44. If V is less than 44, the program executes only the minimum required number of COPY instructions. The COPY sequence includes two tests:

1) Instructions 0080 through 0082 halt the execution of the sequence after V half words have been copied by the sequence; and

2) Instructions 0087 and 0088 ensure that there are no more than 24 COPY executions for each card punched.

After the first card has been punched, and if more information remains to be written, control is passed to instruction 0089. This is the first of three instructions, 0089 through 0091, that test for the coincidence that might occur by having the writing of V half words finish with the punching of a full card. If the coincidence test indicates that additional information remains to be written, the WRITE instruction in location 0092, the resetting instructions in locations 0093 through 0095, and the TRANSFER instruction in location 0096 continue this writing process and return control to the COPY sequence.

After the V half words of the block to be written, together with the full word check sum and the two parameters V and R^1, have been punched, control is transferred to location 0098. Instructions 0098 and 0099 replace the card check sum in location -0120 by the storage check sum. The adding sequence, instructions 0100 through 0110, then reduces this storage check sum by successive addition of the V half words that have been written. After all V half words have been added to the storage check sum, the result is tested for zero by instruction 0111. This instruction also returns control to the sequence which called for the execution of BP 06.

If an error is present in the block of V half words or in the storage check sum, control is not returned to the calling sequence. The calculator then stops at instruction 0112.

The punched-card output of BP 06 may be verified by checking V and R^1 visually and reloading the information by means of SL 05. The successful entry of the information by SL 05 guarantees that the card check sum and the remaining V half words of the block are punched correctly.

Read-Write 02 (RW 02)

This program is designed for reading, writing, adding, reading and adding, or writing and adding U unit records, each consisting of W half words. RW 02 may be used with card machines, tapes, or drums. The U x W half-word locations required in electrostatic storage must have consecutive addresses, and the W half words of each unit record must occupy consecutively-numbered cells. No conversion is included in the execution of this program.

The summing procedure operates on U units records by taking them one at a time. Each unit record must include its own storage check sum. This sum is stored in the two lowest-numbered half-word locations of the block of W required for a unit record.

When RW 02 is called in for reading and adding or writing and adding, the sequence of events is as follows:

1) The first unit record is read into or written from electrostatic storage;

2) This unit record is then summed and zero tested in electrostatic storage;

3) The next unit record is read in or written out, and so on until U unit records have been operated upon.

There are cases in which adding should not accompany reading. For example, consider a set of instructions punched in several binary cards. Normally it would not be convenient to interrupt the sequence of instructions in order to have a check sum for every card, bearing in mind that these check sums are not destroyed by RW 02. If there is not a check sum for each card, the unit record for reading purposes is not the same as the unit record for adding purposes. To deal with this situation RW 02 should first be called in just for reading these cards and then called in a second time for adding the composite unit record which has been introduced into electrostatic storage. Similar remarks apply to writing and adding.

Another case in which the adding operation should be performed independently arises in reading tape backward. In this case RW 02 should first be called in to read the U unit records into consecutively-numbered storage locations in such a way that the check sums appear in the two lowest-numbered cells of each unit record. RW 02 should then be called in a second time to add each of these unit records.

The sequence that calls for the execution of RW 02 must specify the number of unit records (U), the number of half words per unit record (W), the address of the first word in electrostatic storage (R), and, if magnetic drum storage is to be used, the address of the first word in drum storage (S). The signs of U, R, and S should be positive under all circumstances. W should carry a negative sign when tape is being read backward; otherwise W should be positive. W must always be an even integer. S must be zero when drum storage is not to be used.

In addition to U, W, R, and S, the calling sequence must contain an instruction whose operation part determines the function of RW 02. This operation part may be READ, WRITE, READ B, or TR. If a READ or WRITE instruction is positive the reading or writing of each unit record is followed by adding. If either of these instructions is negative, adding is omitted. Since adding should be omitted in the case of reading tape backward, the READ B instruction should always be negative. The address part of the READ, WRITE, or READ B instruction is P, where P is the identification number corresponding to the read-write component to be used. The TR instruction is used to call for adding alone. The address part of this instruction should be 0106 for the example shown.

The calling sequence takes the form:

α		R ADD	α	Basic linkage
α + 1		TR	0048	
	±	READ	P	READ or READ B or WRITE
α + 2	-	READ B	P	or ADD or READ and ADD
	±	WRITE	P	or WRITE and ADD (P =
	+	TR	0106	read-write identification)
α + 3	+	STOP	U	Number of unit records
α + 4	±	STOP	W	Number of half words per unit record
α + 5	+	STOP	R	First electrostatic storage address
α + 6	+	STOP	S	First drum storage address

Note that S must be zero if drums are not being used.

Operation of the Program

The instructions and associated constants occupy half-word locations 0048 through 0127 of electrostatic storage, and full word location -0128 is used in the execution of RW 02. The execution of the program is called for by a transfer of control to location 0048 from a calling sequence of the form described above.

Instructions 0048 through 0073 obtain and store the key instruction and the parameters U, W, R, and S from the calling sequence. The location to which RW 02 is to return control after its own execution is also calculated and stored as the address part of instruction 0123. Instructions 0074 through 0081 prepare the adding sequence for subsequent execution and calculate the indicator words to be used in determining the end of the sequences for adding and copying.

Location 0082 contains the instruction obtained from the calling sequence. The execution of this instruction selects a read-write component if the program is to be used for reading or writing. If RW 02 is to be used for adding only, location 0082 contains the instruction 'TR 0106' which transfers control directly to the adding sequence. The sign of the instruction in location 0082 has no effect upon its execution.

If the program is to be used for reading or writing, instruction 0083 is executed. This instruction specifies the first drum location to be used if the execution of the previous instruction selected a drum. If drum storage is not being used in the execution of

RW 02, the execution of the SET DR instruction has no effect upon any register or storage cell.

The COPY loop includes instructions 0084 through 0090. This loop may be used for reading or writing drums, tapes, or cards. It may also be used for printing without checking. The first word copied is taken from or entered into electrostatic storage location -R. Succeeding full words are taken from or entered into full-word locations -(R + 2), -(R + 4), etc. unless the program is being used for reading tape backward. In this latter case, full-word locations -R, -(R - 2), -(R - 4), are used in the order shown.

Following the execution of the COPY sequence, control is transferred to location 0095. At this point a test is made to determine whether adding is required after copying. If the sign of the instruction obtained from the calling sequence and stored in location 0082 is positive, control is transferred to location 0106.

Instructions 0106 and 0107 obtain the storage check sum of the current unit record and store this check sum in full-word location -0128. The execution of the adding sequence, consisting of instructions 0108 through 0118, is then commenced. This sequence reduces the storage check sum by using the remaining W - 2 half words of the unit record. After W - 2 half words have been added to the storage check sum, this sum is tested for zero by instructions 0119 and 0120.

If the zero test made by instruction 0120 fails, the calculator stops as a result of the execution of the STOP instruction stored in location 0121. Since the address part of this STOP instruction is 0097, the execution of RW 02 may be resumed by pressing the START button.

If the adding sequence and zero test are completed successfully, control is transferred to location 0097. This location may also receive control following the sign test performed by instructions 0095 and 0096 if the instruction stored in location 0082 is negative. This negative instruction signifies that no addition check is to be performed on the unit record being operated upon. Instructions 0097 through 0099 determine whether the program has already dealt with U unit records. If so, control is transferred to location 0122 by the execution of instruction 0099. If the execution of the program is not complete, instructions 0100 through 0104 make the necessary preparations to continue.

When the execution of the program is complete, control is transferred to location 0122 by the execution of the test instruction in location 0099. Instruction 0122 operates to delay the sequence of instruction executions when the MQ register is not free for unrestricted use. This would occur in the case of tape writing. Instruction 0122 has no other function. The execution of instruction 0123 then returns control to the sequence that called for the execution of RW 02.

Self Loading 5 (SL 05)

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
0000	-	COPY	31	[0002]	
0001		R ADD	10	[0003]	
0002	[	ADD	09	0000]	Load SL 05 card
0003	-	COPY	31	[0004]	
0004		STORE A	13	0003	
0005		TR	01	0001	
0006		READ	24	2048	Select card reader
0007	-	COPY	31	0002	Card check sum
0008		TR	01	0010	Not end of file
0009		STOP	00	0001	End of file stop
0010	-	COPY	31	0004	Loading information
0011		R ADD	10	0005	
0012		STORE A	13	0022	Obtain loading address for storage
0013		STORE A	13	0028	check sum and for COPY sequence
0014		ADD	09	0045	
0015		STORE A	13	0033	Initialize ADD sequence
0016		ADD	09	0004	Set up end of COPY index and
0017		STORE A	13	0001	end of ADD index
0018		SUB	05	0045	
0019		STORE A	13	0000	
0020		A RIGHT	23	17	Modify card check sum to give
0021	-	ADD	09	0002	storage check sum
0022	-	STORE	12	[R']	
0023	-	STORE	12	0002	
0024		R ADD	10	0028	
0025		SUB	05	0045	
0026		STORE A	13	0028	
0027		SUB	05	0000	COPY sequence
0028	-	COPY	31	[R+1]	
0029		TR +	03	0024	
0030		TR	01	0033	
0031		READ	24	2048	Select card reader
0032		TR	01	0028	Continue copying
0033		R ADD	10	[R+2]	
0034		STORE	12	0047	
0035		R ADD	10	0033	
0036		ADD	09	0009	
0037		STORE A	13	0033	
0038		SUB	05	0001	ADD sequence
0039	-	LOAD MQ	15	0046	
0040		L LEFT	20	36	mult by 2
0041	-	ADD	09	0002	
0042	-	STORE	12	0002	
0043		TR OV	02	0033	
0044		TR 0	04	0006	Test unit record sum
0045		STOP	00	0002	Error stop
0046		STOP	00	0000	Contribution to unit record sum
0047	[	STOP	00	0000]	

Binary Punching 5 (BP 05)

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
ENTER → 0050		R ADD	10	0047	
0051		STORE A	13	0060	
0052		STORE A	13	0080	Initialize ADD sequence, COPY sequence, COPY index, and ADD index
0053		ADD	09	0048	
0054		STORE A	13	0000	
0055		STORE A	13	0001	
0056		R SUB	06	0049	
0057		SUB	05	0048	Add half word count and loading address to check sum
0058		A RIGHT	23	17	
0059	-	STORE	12	0002	
0060		R ADD	10	[ ]	
0061		STORE	12	0047	
0062	-	R SUB	06	0046	
0063		A LEFT	22	1	
0064	-	ADD	09	0002	ADD sequence for computing card check sum
0065	-	STORE	12	0002	
0066		R SUB	06	0060	
0067		SUB	05	0009	
0068		STORE A	13	0060	
0069		ADD	09	0001	
0070		TR +	03	0060	
0071		WRITE	26	1024	Select card punch
0072	-	COPY	31	0002	Punch nine left
0073	-	COPY	31	0048	Punch nine right
0074		R ADD	10	0080	Initialize end of card index
0075		SUB	05	0004	
0076		STORE A	13	0028	
0077		R ADD	10	0080	
0078		SUB	05	0000	
0079		TR 0	04	0093	
0080	-	COPY	31	[ ]	
0081		R ADD	10	0080	COPY sequence
0082		SUB	05	0045	
0083		STORE A	13	0080	
0084		SUB	05	0028	
0085		TR +	03	0077	
0086		R ADD	10	0080	Coincidence test for end of card and end of COPY conditions
0087		SUB	05	0000	
0088		TR 0	04	0093	
0089		WRITE	26	1024	Select card punch
0090		R ADD	10	0028	Set up end of card index
0091		ADD	09	0073	
0092		TR OV	02	0076	Continue punching
0093		STOP	00	0050	End of punching

Binary Punching 6 (BP 06)

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
ENTER → 0048		ADD	09	0117	
0049		STORE A	13	0056	
0050		ADD	09	0116	
0051		STORE A	13	0060	Preamble
0052		ADD	09	0116	
0053		STORE A	13	0058	
0054		ADD	09	0116	
0055		STORE A	13	0111	
0056		R ADD	10	[ ]	Preserve half word count
0057		STORE	12	0122	
0058		R ADD	10	[ ]	Preserve loading address
0059		STORE	12	0123	
0060		R ADD	10	[ ]	Obtain unloading address for
0061		STORE A	13	0072	storage check sum
0062		STORE A	13	0098	
0063		ADD	09	0117	Initialize COPY and ADD sequence
0064		STORE A	13	0083	
0065		STORE A	13	0100	
0066		ADD	09	0122	Set up end of COPY index and
0067		STORE A	13	0114	end of ADD index
0068		STORE A	13	0056	
0069		R SUB	06	0122	
0070		SUB	05	0123	Modify storage check sum to
0071		A RIGHT	23	17	give card check sum
0072	-	ADD	09	[ ]	
0073	-	STORE	12	0120	
0074		WRITE	26	1024	Select card punch
0075		R ADD	10	0083	
0076		SUB	05	0097	Initialize end of card index
0077		STORE A	13	0113	
0078	-	COPY	31	0120	Image row nine left
0079	-	COPY	31	0122	Image row nine right
0080		R ADD	10	0083	
0081		ADD	09	0114	
0098 ← 0082		TR 0	04	0098	
0083	-	COPY	31	[ ]	
0084		R ADD	10	0083	COPY sequence
0085		SUB	05	0117	
0086		STORE A	13	0083	
0087		ADD	09	0113	
0088		TR +	03	0080	
0089		R ADD	10	0083	Coincidence test for end of card
0090		ADD	09	0114	and end of COPY conditions
0098 ← 0091		TR 0	04	0098	
0092		WRITE	26	1024	Select card punch
0093		R ADD	10	0113	
0094		ADD	09	0112	Step up end of card index
0095		STORE A	13	0113	
0096		TR	01	0083	Continue copying

Binary Punching 6 (BP 06)

LOCATION	INSTRUCTION				REMARKS
	+ / −	OPERATION PART		ADDRESS PART	
0097		STOP	00	0044	Constant for first card
0082 0091 ⇉ 0098		R ADD	10	[ ]	Begin adding record
0099		STORE	12	0120	
0100		R ADD	10	[ ]	
0101		STORE	12	0119	
0102		R ADD	10	0100	
0103		ADD	09	0116	
0104		STORE A	13	0100	
0105		SUB	05	0056	
0106	-	LOAD MQ	15	0118	Adding sequence
0107		L LEFT	20	36	
0108	-	ADD	09	0120	
0109	-	STORE	12	0120	
0110		TR OV	02	0100	
EXIT ← 0111		TR 0	04	[ ]	Exit and zero check
0112		STOP	00	0048	Error stop
0113		COPY	31	[ ]	End of card index
0114		COPY	31	[ ]	End of COPY index
0115		STOP	00	0000	
0116		STOP	00	0001	Necessary constants
0117		STOP	00	0002	

Read-Write 2 (RW 02)

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
ENTER → 0048		ADD	09	0125	
0049		STORE A	13	0060	
0050		ADD	09	0124	
0051		STORE A	13	0062	
0052		ADD	09	0124	
0053		STORE A	13	0064	Set up references to calling sequence
0054		ADD	09	0124	
0055		STORE A	13	0069	
0056		ADD	09	0124	
0057		STORE A	13	0072	
0058		ADD	09	0124	
0059		STORE A	13	0123	
0060		R ADD	10	[ ]	
0061		STORE	12	0082	
0062		R ADD	10	[ ]	
0063		STORE A	13	0091	
0064		R ADD	10	[ ]	
0065		STORE	12	0092	
0066		LOAD MQ	15	0094	Obtain information from calling sequence
0067		L RIGHT	21	0	
0068		STORE MQ	14	0094	
0069		R ADD	10	[ ]	
0070		STORE A	13	0093	
0071		STORE A	13	0084	
0072		R ADD	10	[ ]	
0073		STORE A	13	0083	
0104 → 0074		R ADD	10	0093	
0075		STORE A	13	0106	Prepare adding sequence for next unit record
0076		ADD	09	0125	
0077		STORE A	13	0108	
0078		R ADD	10	0093	
0079		ADD	09	0092	Step up end of record and end of sum indicators
0080		STORE A	13	0093	
0081		STORE A	13	0105	
0082	[			]	READ or WRITE or ADD or TR
0083		SET DR	29	[ ]	Drum address
0084	-	COPY	31	[ ]	
0085		R ADD	10	0084	
0086		SUB	05	0094	
0087		STORE A	13	0084	COPY sequence
0088		ADD	09	0093	
0089		TR 0	04	0095	
0090		TR	01	0084	
0091		STOP	00	[ ]	No. of unit records
0092	[			]	No. of half words
0093		COPY	31	[ ]	COPY indicator
0094	[	STOP	00	0002]	COPY increment
0095		R ADD	10	0082	Is adding required after copying?
0106 ← 0096		TR +	03	0106	

Read-Write 2 (RW 02)

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
0097		R ADD	10	0091	Reduce no. of unit records and test for end of READ, WRITE, ADD procedure
0098		SUB	05	0124	
0099		TR 0	04	0122	
0100		STORE	12	0091	
0101		R ADD	10	0083	Step up initial drum address
0102		ADD	09	0092	
0103		STORE A	13	0083	
0074 ← 0104		TR	01	0074	Repeat
0105		R ADD	10	[ ]	End of sum indicator
0096 → 0106	-	R ADD	10	[ ]	Check sum
0107	-	STORE	12	0128	
0108		R ADD	10	[ ]	Adding sequence
0109		STORE	12	0127	
0110	-	R ADD	10	0126	
0111		A LEFT	22	1	
0112	-	ADD	09	0128	
0113	-	STORE	12	0128	
0114		R SUB	06	0108	
0115		SUB	05	0124	
0116		STORE A	13	0108	
0117		ADD	09	0105	
0118		TR +	03	0108	
0119	-	R ADD	10	0128	Is unit record sum equal to zero?
0120		TR 0	04	0097	
0121		STOP	00	0097	Stop if not zero
0122		WRITE	26	2052	Delay if necessary
EXIT ← 0123		TR	01	[ ]	Return
0124		STOP	00	0001	Necessary constants
0125		STOP	00	0002	
0126		STOP	00	0000	
0127	[			]	

INTERNATIONAL BUSINESS MACHINES CORPORATION
590 Madison Avenue, New York 22, N. Y.

Form No. 32-9436-0 (823CM)

Kolsky

[redacted]

BULLETIN 2

00039

PROGRAMMING CONVENTIONS

for the

IBM

TRADE MARK

ELECTRONIC

DATA PROCESSING MACHINES

TYPE 701 AND ASSOCIATED EQUIPMENT

INTERNATIONAL BUSINESS MACHINES CORPORATION

APPLIED SCIENCE DEPARTMENT

New York, New York

The following information is
not final and is subject to mod-
ification. The information is
the best available at this time.
August, 1952

Conventions

A portion of memory capable of retaining a word is termed a 'location' or a 'cell'. As mentioned in the Preliminary Report on the IBM Electronic Data Processing Machines, Type 701, each cell (or location) is identified by an address which is regarded as an integer. Consequently, the twelve bits of the address part of an instruction are interpreted to be an integer. The five bits of the operation part of an instruction are also regarded as an integer. With the address part represented by an integer y where

$$0 \leq y < 2^{12}$$

and the operation part represented by an integer x where

$$0 \leq x < 2^5$$

the entire instruction is represented by an integer i

$$i = \pm (x \cdot 2^{12} + y)$$

Thus, when modifying instructions by simple additions the point lies between accumulator positions 17 and 18.

It is convenient to express addresses as decimal integers since the decimal system is most familiar, and for ease in reading programs, alphabetic abbreviations are used as operation parts, e.g.,

- ADD 1492

Of course, the addresses expressed in decimal and the operation abbreviations must be converted to binary information before interpretation by the 701. Decimal to binary conversion is readily accomplished by the 701 itself and is discussed in a subsequent bulletin.

The programming form and storage layout charts now used by the Applied Science Mathematical Committee are shown in the latter pages. In the programming form, the column between the Operation and Address parts is used for the decimal operation code, i.e.

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
+1066	-	R ADD	10	1492	Place the contents of cell -1492 in the accumulator

The location, the sign, the operation code and the address are subsequently punched in cards for loading decimal instructions. The storage charts are convenient for planning the arrangement of programs and data in Electrostatic Storage.

Since, in describing a program, it is frequently necessary to refer to a particular cell (or location) in memory, the phrase 'cell α' is used to mean 'the cell whose address is the integer α'. Thus, the phrase '1/10 is retained in cell -1492' means '1/10 is retained in the cell identified by the address -1492'. This type of phrase frequently will be further abbreviated to

$$L(x) = \alpha$$

by which is meant 'the location of the quantity x in memory is the cell whose address is α', e.g.

$$L\ (1/10) = -1492$$

Similarly, it is frequently necessary to refer to the quantity or word retained in a particular location. The quantity or word retained is usually referred to as 'the contents of the cell', e.g. '1/10 is the contents of cell -1492'. The notation

$$C\ (\alpha) = x$$

means 'the contents of cell α is the quantity x,' e.g.

$$C\ (-1492) = 1/10$$

C and L are reciprocal symbols.

Instructions and comments may be written using this notation:

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
1066	-	R ADD	10	L(1/10)	Place 1/10 in the accumulator
1067	-	STORE	12	1494	C(-1494) is replaced by 1/10

It should be noted that in the above program the actual array of binary digits that represent 1/10 is not precisely specified, since position of the point is not specified. However, the use of the L and C symbols in connection with instructions or instruction modification is precise because of the convention to regard instructions as integers. Thus, in the program,

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
1066	+	R ADD	10	1068	Modify the copy instruction
1067	+	ADD	09	L(2)	
1068	-	COPY	31	[0000]	
1069	+	STORE A	13	1068	
1070	+	TR	01	1067	
1071					

the execution of instruction 1067 adds the integer 2 to the instruction in the accumulator, or equivalently, a binary 1 to accumulator position 16.

When an instruction or address part is modified by computation, the computed instruction or instruction part is enclosed by square brackets. See instruction 1068 above.

Arrows are drawn to the left of the instructions to indicate the action of the transfer instructions, as in the program above. Dotted arrows are drawn to indicate the action of COPY or SENSE skips.

For simple exposition it is frequently not convenient to write actual addresses, as 1492 or 1066 above. A symbol, such as a Greek or alphabetic letter may also be employed to represent an address where the correspondence to a particular address is arbitrary or not yet assigned. For example, consider the following program

LOCATION	INSTRUCTION				REMARKS
	+ −	OPERATION PART		ADDRESS PART	
α	-	R ADD	10	1492	Duplicate the contents of cell -1492
$\alpha + 1$	-	STORE	12	1494	in cell -1494

With this notation it is possible to refer to the instruction α (i.e., the instruction stored in some cell α) without actually specifying where the program is to be located in memory. Of course, Greek alphabetic characters may also be used for the address part of an instruction.

A subscript is used to indicate the base of a number when the context is not clear:

$$(1101)_2 = (15)_8 = (13)_{10}$$

The IBM card presently used for recording binary information is shown on the last page. Identification is punched in the first eight card columns. In text involving card input the following notation has been adopted. The word obtained by the first COPY execution is designated by 9_L to indicate the left portion of the 9's row. The word obtained by the second COPY execution is designated by 9_R, and so on, the last word of the card image being designated by 12_R for the right portion of the 12's row.

Branching

The execution of the conditional transfer instruction

```
α        TR +   β
```

will be followed by the execution of either the instruction in cell β or the instruction in cell $\alpha + 1$. Instruction β will be executed if the Accumulator is positive and instruction $\alpha + 1$ will be executed if the Accumulator is negative. Thus the conditional transfer operation provides the possibility of executing either the instructions in cells β, $\beta + 1$, $\beta + 2$, ... or the instructions in cells $\alpha + 1$, $\alpha + 2$, $\alpha + 3$, ..., the choice based on the condition of the Accumulator. The sequence of instructions in cells β, $\beta + 1$, $\beta + 2$ ... is called a branch, as is the sequence of instructions in cells $\alpha + 1$, $\alpha + 2$, ... Each branch of a program is ordinarily a different computational procedure. For example, the instructions in cells α, $\alpha + 1$, $\alpha + 2$, ... might be designed to evaluate $y = \sin x$ for x in the interval 0 to $\pi/2$, and the other branch, the instructions in cells β, $\beta + 1$, $\beta + 2$, ..., designed to evaluate the same function for x in the interval $\pi/2$ to π. The conditional transfer instruction could be used to select the appropriate branch for any argument x in the interval 0 to π. That part of a program that provides the possibility of executing one of several branches is called a fork. A conditional transfer instruction is a fork. However, a fork may be more than one instruction. Consider the pair of instructions

```
α          TR 0     β
α + 1      TR +     γ
```

This pair of instructions is a three-branch fork. The instruction in cell β will be executed if the number in the Accumulator is zero. Instruction γ will be executed if the content of the Accumulator is positive and *not* zero. Instruction $\alpha + 2$ will be executed if the Accumulator content is negative and *not* zero. If γ is set equal to β, then the branch of instructions starting at cell β will be executed if the number in the Accumulator is positive *or* zero. By appropriately specifying the address parts of this pair of instructions *any* of the conditions equal to, greater than, less than, greater than or equal to, or less than or equal to may be realized.

In the preceding example the choice of the branch to be executed is determined by the condition of the Accumulator. The following four-branch fork illustrates that this is not always the case. Let a, b, c, and d be the addresses of the first instruction of four branches. Suppose the criterion for the branch selection is a parameter Δ which may take on the values 0, 1, 2, and 3. For $\Delta = 0$ the instruction in cell a is to be executed, for $\Delta = 1$, instruction b is to be executed, and so on. The selection may be accomplished by computing the address part of an unconditional transfer instruction.

```
α            R ADD       L(Δ)
α + 1        ADD         L(α + 4)
α + 2        STORE A     α + 3
α + 3        TR          [α + Δ + 4]
α + 4        TR          a
α + 5        TR          b
α + 6        TR          c
α + 7        TR          d
```

The address part of instruction $\alpha + 3$ is computed by instructions α, $\alpha + 1$ and $\alpha + 2$. Instruction $\alpha + 3$ provides the selection of instruction $\alpha + \Delta + 4$ which is an unconditional transfer to the appropriate instruction a, b, c or d.

It is occasionally convenient not to store the first three instructions of this fork adjacent to the unconditional transfer instructions, but rather to separate the sets of instructions:

```
α                R ADD          L(Δ)
α + 1            ADD            L(β + 1)
α + 2            STORE A        β
.
.
.
β                TR             [β + Δ + 1]
β + 1            TR             a
β + 2            TR             b
β + 3            TR             c
β + 4            TR             d
```

Calculations may then be performed after the selection of the branch but before executing the selected branch. This fork is then called a preset fork.

The conditional transfer instructions may also have computed address parts and consequently provide multi-branch forks.

Basic Linkage

Suppose that in the execution of program A two quantities x and y are computed and that it is desired to evaluate

$$z = \sqrt{x} + \sqrt{y}$$

Let program B be a program for the calculation of square roots. In order to evaluate z, program B must be executed twice, first to calculate $\sqrt{x}$ and then to calculate $\sqrt{y}$. Program B must first be supplied with x and then executed, and subsequently supplied with y and then executed. In the evaluation of $\sqrt{x}$ the last executed instruction of program B must transfer to the instructions for supplying y. In the evaluation of $\sqrt{y}$ the last executed instruction of B must transfer to the instructions for adding the results to form z. Thus the last executed instruction of program B must be a fork in order to provide the different computational procedures. A usable technique is to appropriately modify the last instruction of program B before each execution of the program (this would be a preset fork). An alternative method is to supply program B with information through which the fork may be set, that is, supply the address of the instruction to be executed after program B. In this latter technique two quantities are required by program B--the quantity x and the 'return address'.

Consider the following simpler problem. Let two programs A and B be given, where program B is to be executed after the instruction of program A in cell $\alpha - 1$, and the execution of program A is to be continued after the execution of program B. Let the instruction stored in cell β be the first instruction of program B. Suppose that program B requires no information other than the address of the instruction to be executed after its completion. To provide information to compute this address two instructions are stored at locations α and $\alpha + 1$.

```
α - 1                  ....
α          R ADD       α
α + 1      TR          β
α + 2                  ....
```

The instruction in cell $\alpha + 2$ is to be executed after the completion of program B. Instruction α places the numerical representation of R ADD α in the accumulator. The contents of the accumulator is not affected by the execution of instruction $\alpha + 1$.

```
β          ADD       L (2)
β + 1      STORE A   β + k
...
β + k      TR        [α + 2]
```

The address $\alpha + 2$ is calculated by adding 2 to the contents of the accumulator through the instruction β. Instruction $\beta + 1$ stores $\alpha + 2$ in the address part of instruction $\beta + k$. It is immaterial that after the execution of instruction β the accumulator also contains the operation part R ADD, since only the portion of the accumulator containing $\alpha + 2$ is stored, by instruction $\beta + 1$, in the address part of instruction $\beta + k$. Then the execution of instruction $\alpha + 2$ of program A will follow the execution of instruction $\beta + k$.

Using this technique the programmer is required to know only the location of the first instruction of program B. Note that instructions β and $\beta + 1$ could have been placed between instructions α and $\alpha + 1$. However, this would require writing two additional instructions in program A for each time it is desired to execute program B.

An alternative for instructions β and $\beta + 1$ is

```
β          ADD       L (-9·2^12 + 2)
β + 1      STORE     β + k
```

since before the execution of instruction β the accumulator contains the instruction

R ADD α

or equivalently, $10 \cdot 2^{12} + \alpha$, and the execution of instruction β yields

$$1 \cdot 2^{12} + (\alpha + 2)$$

or equivalently, the instruction

TR $\alpha + 2$

Suppose that program B requires additional information, such as the quantity x in a program for the calculation of $\sqrt{x}$. If such quantities are computed in program A, they can be stored as they are computed in the cells reserved for them in program B. If only one quantity is to be exchanged, the Multiplier-Quotient register is frequently convenient.

Occasionally, some of the information program B requires is not computed. For example, information may be specified by the programmer, such as the number of times program B is to be repeated, or the address of a quantity to be used by program B. Let θ be a half-word of such information. θ can be stored in program A in the cell $\alpha + 2$.

```
α + 1      ....
α          R ADD     α
α + 1      TR        β
α + 2                θ
α + 3      ....
```

Instruction $\alpha + 3$ is to be executed after the completion of program B. In order for program B to use the quantity θ it must be supplied with the location of θ. Suppose instruction $\beta + j$, of program B, refers to θ; that is, the address part of instruction $\beta + j$ is to be $\alpha + 2$, the location of θ. The first four instructions of program B now become

```
β          ADD       L (2)
β + 1      STORE A   β + j
β + 2      ADD       L (1)
β + 3      STORE A   β + k
```

Instructions β and $\beta + 1$ compute $\alpha + 2$ and store the quantity as the address part of instruction $\beta + j$. Instructions $\beta + 2$ and $\beta + 3$ compute $\alpha + 3$ and store that quantity as the address part of the last instruction of program B. This last instruction to be executed in program B is again a transfer instruction which returns control to program A.

In a similar manner n half-words of information may be exchanged.

The linkages described above result in unconditional entry to program B after the execution of instruction $\alpha + 1$ of program A. Conditional entry is readily obtained by using the Multiplier-Quotient Register to retain the address α:

```
α - 1       . . .
α           LOAD MQ      α
α + 1       TR 0         β
α + 2       . . .

β           L LEFT       35
β + 1       ADD          L (2)
β + 2       STORE A      β + k
 . . .      . . .
β + k       TR           [α + 2]
```