EUR 2637.e
VOL. III

EUROPEAN ATOMIC ENERGY COMMUNITY - EURATOM

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090

VOL. III : THE SIMULATOR PROGRAM

by

A.F.R. BROWN

1966

Joint Nuclear Research Center
Ispra Establishment - Italy

Scientific Information Processing Center - CETIS

This report is on sale at the addresses listed on cover page 4

at the price of FF 7,—	FB 70,—	DM 5,60	Lit. 870	Fl. 5,10

When ordering, please quote the EUR number and the title, which are indicated on the cover of each report.

Printed by Guyot, s.a.
Brussels, January 1966

EUR 2637.e
VOL. III

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090
VOL. III : THE SIMULATOR PROGRAM by A.F.R. BROWN

European Atomic Energy Community - EURATOM
Joint Nuclear Research Center
Ispra Establishment (Italy)
Scientific Information Processing Center - CETIS
Brussels, January 1966 - 60 Pages - FB 70

In the field of non-numerical data processing it is often more profitable to use a medium-size computer instead of a big one. Compilation, however, may better be done on a bigger machine.
The four volumes of this report describe a symbolic programming language, its compiler for the IBM 7090 which produces IBM 1401 object programs, and a simulator permitting the execution of these programs on the IBM 7090.

EUR 2637.e
VOL. III

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090
VOL. III : THE SIMULATOR PROGRAM by A.F.R. BROWN

European Atomic Energy Community - EURATOM
Joint Nuclear Research Center
Ispra Establishment (Italy)
Scientific Information Processing Center - CETIS
Brussels, January 1966 - 60 Pages - FB 70

In the field of non-numerical data processing it is often more profitable to use a medium-size computer instead of a big one. Compilation, however, may better be done on a bigger machine.
The four volumes of this report describe a symbolic programming language, its compiler for the IBM 7090 which produces IBM 1401 object programs, and a simulator permitting the execution of these programs on the IBM 7090.

EUR 2637.e
VOL. III

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090
VOL. III : THE SIMULATOR PROGRAM by A.F.R. BROWN

European Atomic Energy Community - EURATOM
Joint Nuclear Research Center
Ispra Establishment (Italy)
Scientific Information Processing Center - CETIS
Brussels, January 1966 - 60 Pages - FB 70

In the field of non-numerical data processing it is often more profitable to use a medium-size computer instead of a big one. Compilation, however, may better be done on a bigger machine.
The four volumes of this report describe a symbolic programming language, its compiler for the IBM 7090 which produces IBM 1401 object programs, and a simulator permitting the execution of these programs on the IBM 7090.

On sale at
PRESSES ACADEMIQUES EUROPEENNES
93, Chaussée de Charleroi, Bruxelles 6

EUR 2637.e
VOL. III

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090
VOL. III : THE SIMULATOR PROGRAM by A.F.R. BROWN

European Atomic Energy Community - EURATOM
Joint Nuclear Research Center
Ispra Establishment (Italy)
Scientific Information Processing Center - CETIS
Brussels, January 1966 - 60 Pages - FB 70

In the field of non-numerical data processing it is often more profitable to use a medium-size computer instead of a big one. Compilation, however, may better be done on a bigger machine.
The four volumes of this report describe a symbolic programming language, its compiler for the IBM 7090 which produces IBM 1401 object programs, and a simulator permitting the execution of these programs on the IBM 7090.

This volume is concerned with the simulator program. The description of this program takes the form of comments on the corresponding flow charts given in the fourth volume. Futhermore, the use of the program is explained.

This volume is concerned with the simulator program. The description of this program takes the form of comments on the corresponding flow charts given in the fourth volume. Futhermore, the use of the program is explained.

This volume is concerned with the simulator program. The description of this program takes the form of comments on the corresponding flow charts given in the fourth volume. Futhermore, the use of the program is explained

This volume is concerned with the simulator program. The description of this program takes the form of comments on the corresponding flow charts given in the fourth volume. Futhermore, the use of the program is explained.

EUR 2637.e
VOL. III

EUROPEAN ATOMIC ENERGY COMMUNITY - EURATOM

THE COMPILATION AND PROCESSING OF IBM 1401 PROGRAMS ON IBM 7090

VOL. III : THE SIMULATOR PROGRAM

by

A.F.R. BROWN

1966

Joint Nuclear Research Center
Ispra Establishment - Italy

Scientific Information Processing Center - CETIS

TABLE OF CONTENTS

Introduction

For test purposes or for short production runs it is very useful to have the possibility of executing a program immediately after its compilation. Therefore in addition to the IBM 7090/1401 compiler system described in volumes 1 and 2 a simulator system has been developed. It permits execution of IBM 1401 programs on the IBM 7090.

The first 49 pages of this volume give a description of this simulator system; continuous reference is made to the corresponding flow charts. These flow charts are printed on pp. 68-99 of the fourth volume.

The second part of the present volume is a manual for the practical use of the simulator system.

S U M M A R Y

In the field of non-numerical data processing it is often more profitable to use a medium-size computer instead of a big one. Compilation, however, may better be done on a bigger machine.

The four volumes of this report describe a symbolic programming language, its compiler for the IBM 7090 which produces IBM 1401 object programs, and a simulator permitting the execution of these programs on the IBM 7090.

This volume is concerned with the simulator program. The description of this program takes the form of comments on the corresponding flow charts given in the fourth volume. Furthermore, the use of the program is explained.

Every character of the 1401 memory is represented by a word in the 7090 memory, from MEM to MEM+7999. The word mark of the 1401 character is represented by the sign bit of the 7090 word, the zone bits by the two low-order bits of the 7090 word decrement, and the numerical bits by the four low-order bits of the 7090 word address. Thus, for example, a group-mark with word-mark is represented by (octal) 400003000017; a blank with a word mark by 400000000000; the letter A with no word mark by 000003000001; the letter K with a word mark by 400002000002; the letter T with no word mark by 000001000003; and an equal sign with a word mark by 400000000013.

First of all, the simulation program puts octal 400003000017 in the accumulator and performs ANS on cells MEM through MEM+7999, in order to give a "1401 memory" containing characters that are random, but compatible with this representation (see BEGIN to BEGA in the listing.) Then the data file begins to be read from the normal monitor input tape. This may begin with from 0 to any number of BCD cards having ****** in cols. 73-78. These are for patching the 7090 program itself. A patch card contains four fields, cols. 1-18, 19-36, 37-54, and 55-72. Let the 18 characters in a field be represented KLLLLLPDDDDDTAAAAA. LLLLL is the octal address, relative to the beginning of the program, into which the patch word represented by octal PDDDDDTAAAAA is to be stored. If K=0, the decrement DDDDD and address AAAAA will be unchanged. If K is a minus sign, the address will be increased by the absolute address of the starting location of the program, before the patch word is stored. If K is a plus sign,

Manuscript received on October 27, 1965.

the decrement DDDDD will be so increased, and if K is a blank, both the address and the decrement will be so increased. (See BEGA through BEGD in the listing.)

The first card in the data file that does not have ****** in cols. 73-78 is the control card, with control characters punched beginning with column 1, and continuing up to but not including the leftmost blank. At least one of cols. 1-72 must contain a blank, to terminate the reading of the control characters. See Flow Chart 1 for the interpretation of this card. The sequence of the control characters has no significance, except that if the character next after a 1,2,3,4,5 or 6 is not H or L, the program will behave as if an H were inserted immediately after the digit, and before the character that actually follows it. The significance of the control characters is as follows:

& The 1401 program will behave as if the I/O check stop switch were on. If this character does not appear on the control card, it will behave as if the switch were off. The only practical significance of this has to do with the reading of cards from the input file by the 1401 program. If the "look-ahead" from the card last read shows that the next card will be BCD, it can be read by the 1401 program in either BCD or binary form. But if the next card will be binary, and the 1401 program tries to read it in the BCD mode, the simulation program will put into positions 1-80 of the 1401 memory exactly what th, real 1401 would put. Since a card in the monitor input is treated as binary only if it has a 7-9 punch in column 1, such a card would cause a validity check if read physically by a real 1401. If the I/O check stop switch is on, this will cause the 7090 program to dump the "1401 memory" and exit. If the switch is off, the simulation program will continue normally.

A,B,C,D,E,F,or G The simulation program will behave as if the corresponding sense switch on the 1401 console were on. For each of these letters that does not appear on the control card, the program will behave as if the corresponding sense switch were off.

H Logical switch "HALTS" (located in cell CONSW+7, immediately after the cells for the seven sense switches) is set on. If this switch is off, the first "HALT" instruction obeyed by the simulated 1401 will cause the simulator to output the message "1401*HALT", dump the 1401 memory, and exit. If the switch is on, each 1401 "HALT" will cause the 7090 to print two lines on the on-line printer, and then halt. The first line will show the contents of the simulated 1401 I-address, A-address, and B-address registers. The second line will contain the message to the operator: "PROGRAMMED 1401 HALT. PRESS START TO CONTINUE. TURN ON SIGN KEY AND START TO END 1401 JOB." The 7090 operator can have been instructed to check the contents of the 1401 address registers as printed, and continue or terminate the 1401 job accordingly. If the job is terminated, a 1401 memory dump will be output before the simulation program exits.

T Logical switch "TPST" is set on. This means that the 1401 program will be initiated by the instructions /080 M(U1001R ,001 B001 as if the "LOAD TAPE" button were pressed. If this switch is off, because T has not appeared among the control characters, the program will be initiated by /080 ,001 1001 as if the "LOAD" button on the card reader were pressed.

U Logical switch "UDISC" is set on. This means that the 1401 will "read cards" from the disc unit. If this switch is off, the 1401 will "read cards" from the normal monitor input tape. The first card to be read will be the one that immediately follows the control card in the input file. In order to enable the simulator to recognize the "last card read" condition in the card reader, a card with this punched in columns 1-72:
///// THE NEXT CARD WOULD HAVE BEEN PHYSICALLY LAST IN THE 1401 INPUT.
must be inserted in the input file immediately BEFORE the last card.

If the 1401 is to "read cards" from the disc unit, this will be because the simulator is being used to try out a program which has just been compiled by the special 7090 compiler of 1401 programs. In that case, no tapes are to be mounted before, or removed after, the 1401 job. The description of the compiler will explain this.

$ Logical switch "CDUMP" is set. This means that before the simulator terminates the 1401 job, it will output, for punching, 101 extra cards. The first will have the letters MEMORY punched on it so as to be recognizable on sight, and the other 100 cards will each contain the information from 80 cells of 1401 memory - cells 0 to 79 on the first of these cards, 80 to 159 on the second, and so on. The 80 characters are contained in binary form on the upper half of the card. In the lower half, the eighty word marks from the same 1401 cells are represented, as a blank for no word mark and a minus sign for a word mark. The leftmost column of the lower half will contain 7 and 9 punches as well, so that an absence of word mark in the corresponding cell appears as "5", and a word mark as "N".

The program for this punching out begins at ENDD and ends just before PAGE. Refer to flow chart 50.

- Logical switch "REMEM" is set on. This means that the 1401 program will not be initiated in the normal way, with the simulation of the LOAD button. Instead, the reading of binary cards will be simulated. These cards are supposed to be the contents of memory at the end of a previous run, whose punching is described in the preceding section. The first binary card is used to fill 1401 cells 0 to 79, the second 80 to 159, and so on.

There need not be a full hundred of these binary cards; if there are only fifty, they will supply the contents of cells 0 to 3999 , and so for any other number of them. The series of binary cards is ended by a BCD card. If column 1 of this card is blank, a normal program load is then carried out; i.e. the simulation of the LOAD button on the card reader or the LOAD TAPE button, depending on whether switch "TPST" is off or on. But if column 1 is not blank, it is to be the leftmost column of a left-justified decimal number between 1 and 7999; the simulator will execute a branch to that 1401 address and begin the program from there.

For an example of how control cards with $ and = might be used, consider the 1401 SPS assembly program, in the version that does not use tapes. One loads the first-stage program into the card reader, followed by the symbolic cards that have to be assembled. The first-stage program reads the symbolic cards and punches out some cards. It is then necessary to load the second-stage program into the card reader, put the cards that were just punched out behind it, and press LOAD. The second-stage program does not, however, clear memory before loading itself in, as it needs to use some tables that have been left behind by the first-stage program. After the second-stage program, each stage of the assembly can be carried out separately without any need to preserve memory contents.

In simulating such an assembly, one would put the following in the monitor input:

(a) an ID card

(b) the binary program deck of the simulator

(c) an end-of-file card (7-8)

(d) a simulator control card with A$ punched in columns 1-2, indicating that the program was to be run with sense switch A on, and that the contents of memory were to be punched out when the program stopped. The absence of H on the control card would indicate that the first 1401 halt would end the 7090 job.

(e) the first section of the SPS assembly program on cards, followed by the symbolic program cards to be assembled. The last symbolic card would be preceded by an extra card, with:

//////THE NEXT CARD WOULD HAVE BEEN PHYSICALLY LAST IN THE 1401 INPUT. punched in cols. 1-72.

In the monitor output from this run, one would get a printed 1401 memory dump, and a series of punched cards consisting of:

(a) the ID card

(b) the cards that would have been punched if the program had been run on the 1401.

(c) a signal card, with MEMORY punched visibly on it

(d) 100 cards containing the 1401 memory, to be saved

(e) the end-of-file and OK cards.

To do the second stage of the SPS assembly, it would be necessary to make another passage through the 7090 monitor, with the following cards:

(a) an ID card

(b) the binary program deck of the simulator

(c) an end-of-file card

(d) a simulator control card with A= punched in columns 1-2

(e) the 100 cards from (d) in the preceding output

(f) a blank card, showing that the start for the present stage will be with the LOAD button

(g) the 1401 program cards for the second stage of the SPS assembly

(h) the cards from (b) in the preceding output with an extra card containing "//////THE NEXT CARD etc." inserted before the last one

(i) an end-of-file card.

If it were necessary to begin the second stage by pressing the START button instead of the LOAD button, one would look at the memory dump given by the first stage and note the I-address given after IREG on the second line. This number would be punched in columns 1 ff. of the card at (f) in the input to the second stage.

The loading of these memory cards is programmed from BGA to BGV in the simulator, and forms part of flow chart 2.

(After this digression, we continue the list of control card characters.)

A DIGIT : This is supposed to name a tape that will be used by the 1401 program. If the digit is 7, 8, 9, or 0, or if the digit has already appeared on the control card, this is an error, and the 7090 job will be ended. Only digits 1 to 6 are valid as 1401 tape addresses, and as there is no reason for repeating a digit, a repetition is assumed to be an error. These two errors are shown by two branches to ERROR on the right side of flow chart 1. The program for ERROR is shown on flow chart 50.

There are many branches to ERROR in the flow charts, corresponding to branches to ERRA, ERRB, ERRC and so on in the simulator programs. Each of the latter branches causes the printing of a message, followed by a memory dump. The dump begins at location DUMP in the simulator program, but there is no location ERROR.

The message for the present case is: CONTROL.CARD.NAMES.TAPE. 7890.OR.REPEATS.A.TAPE

The digit is to be followed immediately on the control card by either H or L, indicating the density in which the tape is to be used. If the next character is not H or L, the simulator proceeds as if it were followed by H, and the character which follows the number is treated as a normal control character.

1401 tapes 1, 2, 3, 4, 5, and 6 are represented by 7090 tapes A-5, B-5, A-6, B-6, A-7, and B-7 respectively, and it is up to the user to arrange that if necessary, the appropriate tapes will be mounted on the 7090 before the job, and dismounted afterwards.

*   : Logical switch "TRACE" is set on. This means that before every execution of a 1401 instruction by the simulator, the 1401 address of its leftmost character will be output on the monitor listing tape. These addresses will be arranged in columns of 50, 20 columns per page. Such pages will appear in the output mixed with the pages of simulated 1401 printer output.

The recording of each address is done by INS&2 through TRB-1 in the simulator program, shown in flow chart 17.

The output of each page is done by sub-routine TRD in the simulator program, shown as flow chart 18.

K   : Logical switch "SWICK" is set on. This means that an error message and termination of the job will take place whenever an instruction containing an invalid A- or B-address begins to be executed. If "SWICK" is off, however, an error will not occur because of an invalid address in a "no-op" instruction, or because of an invalid A-address in a conditional branch instruction if the condition is not met and the branch does not occur.

In these cases, an invalid address is put into the corresponding address register as 8000; this will cause an error exit at a point below INS in flow chart 17, if the condition is met and the branch occurs. Surprisingly, the 8K 1401 at CETIS behaves more nearly like the simulator with switch SWICK off. But if it is modified in future so as to be more stringent, it could be simulated with SWICK on.

After the control card has been read and interpreted, the simulator goes to BEGIB, which is the normal exit from flow chart 1, and the beginning of flow chart 2. If any digits appeared on the control card, logical switch TMESG is on, showing that tapes will be used. If U has appeared on the control card, logical switch UDISC is on, showing that the job was immediately preceded by a compilation of a 1401 program by the 7090, and that the card input is to be taken from discs. In that case, it is assumed that the simulator is carrying out a trial, in which some tapes may be written and then read back, but no pre-existing tapes are to be mounted for reading, and no tapes that may be written will be dismounted for saving. Since the program is being tried for the first time, it would be inappropriate to slow up the monitor by tape handling. But if there are tapes to be used, and switch UDISC is off, the simulator stops the 7090, to allow the operator to mount any necessary tapes. Pressing START on the 7090 causes the simulator to proceed in flow chart 2.

Now switch REMEM is tested, and if it is on, cards are read to restore the memory contents from some earlier run, as described above for $ and = on the control card.

Then 1401 program execution begins, either by branching to a starting address given by the card that follows the memory cards (see the bottom of flow chart 2) or by branching to BEGET in the simulator program, where either LOAD CARD or LOAD TAPE 1 is simulated.

Flow chart 2 uses simulator subroutines RDKB (read a binary card from disc - see flow chart 5), RDKD (read a BCD card from disc - see flow chart 4) and RCD (carry out the 1401 operation of reading a card in the BCD mode into locations 1 to 80 - see flow chart 10).

The error exits on flow chart 2 go to IER, where the message EOF*AFTER*CONTROL*CARD is printed before the dump.

Flow chart 3 shows simulator subroutine RDKM, which obtains either a whole BCD card or half a binary card from the discs.

Flow charts 4 and 5 show simulator subroutines RDKD and RDKB.

Flow chart 6 shows simulator subroutines NPD and NPB, which are used by subroutines RCG and RCE respectively, for converting a 7090 record (BCD or binary respectively) into a 1401 record, without disturbing word marks in the 1401 cells.

Since character table RDTAB is used by NPD, we may insert here a description of the character tables in the simulator program. Let us use "z" to represent the two-bit number contained by the zone bits of a 1401 character under consideration, and "n" to represent the four-bit number contained by its numerical bits.
CTAB : This table is used only for simulating the 1401 "compare" instruction. Given a 1401 character, one finds at CTAB&16z&n a number, between 0 and 63, which represents its position in the ascending list of characters for 1401 comparison. After the number, the character itself is put as a comment in the simulator program

listing. For instance, at CTAB&9, one finds "PZE 63 9", showing that the character 9 has the highest possible value (63) for 1401 comparison.

WDTAB : This table is used in forming records for writing out on tape in the BCD mode. Given a 1401 character, one will find at WDTAB&16z&n the corresponding 7090 character as it would be stored in memory after reading from a BCD tape or before writing on a BCD tape. For instance, the 1401 blank has all zone and numerical bits 0, so WDTAB&16z&n = WDTAB, and at WDTAB the table contains "OCT 60", and 60 is the octal equivalent of blank in the 7090. The 1401 character with an A-zone bit=1, and all other bits=0 gives 16z&n=16, and at WDTAB&16 the table contains "OCT 60", because this character would be written on BCD tape by the 1401 as a blank.

No table is needed for forming a record to be written in the binary mode. It is only necessary to put the 1401 zone and numerical bits next to each other to get the character which the 7090 will write out in the same way as if a 1401 had written it out in the binary mode.

RMARK is given as label to the location in table WDTAB that is used in writing out the record mark. This has to be altered from octal 72 to octal 53 during the simulation of 1401 printing. Since the simulator has to write an output tape that is compatible with the standard monitor listing program, a record mark must not be put in the middle of a line, where it would cause the listing program to break off the line. So record marks are replaced by dollar signs for this purpose.

RDTAB : This table is used in breaking up records that have been read in the BCD mode by the 7090 into 1401 characters.

When the 7090 reads a zero from tape, it puts octal 00 in the memory, and at RDTAB&0, the table contains "PZE 10" ("PZE 10,,0" would be more explicit) showing that the corresponding 1401 character has a numerical part equal to 10, and a zone part equal to 0. When the 7090 reads a blank from tape, it puts octal 60 in the memory, and at RDTAB&48, the table contains "PZE 0", showing that the corresponding 1401 character has all bits equal to 0.

No table is needed for a record that has been read in the binary mode, since the 7090 and 1401 read such a tape almost identically. It is enough to make the leftmost two bits of the 7090 character into the zone bits of the 1401 character, and its rightmost four bits into the numerical bits.

Flow chart 7 shows the simulator subroutines RCG (read a BCD card and store its contents in 1401 cells 1-80 without disturbing word marks) and RCE (read a binary card into 7090 cells N to N&79 and N&100 to N&179, without disturbing sign bits. If N=MEM&401, this corresponds to the reading of a binary card by the 1401.)

Note that when an end-of-file is read on the input tape, this does not correspond to an end to the deck of cards being read by the 1401. An end-of-file corresponds to a 7-8 card in the monitor input, which would be physically read by the 1401 as a tape mark in cell 1, and blanks in cells 2-80.

It is necessary to allow RCE to read into a part of the 7090 memory outside the simulated 1401 memory because a binary card can be read by a decimal card read instruction if the I/O check stop switch is off. The simulator will read the corresponding binary record from tape; put the 160 characters in a workspace, and then deduce what the 1401 would have put in cells 1-80. When a binary card is properly read by the 1401 program, the 160 characters go into 1401 cells 401-480 and 501-580, and then the deduction for cells 1-80 is made.

When RCG reads a card containing "//////THE NEXT CARD WOULD HAVE BEEN etc." it sets switch LASTC on, and makes a special exit. RCG is used only by RCD and RCB, and when either of them gets the special return from RCG, it immediately re-enters RCG to read the "physically last" card.

Flow chart 8 shows subroutine RCF, which converts a binary card image in 1401 form into the equivalent BCD card image in 1401 memory cells 1-80. It is used by RCD, for reading a binary card as if decimal, with the I/O check stop switch off, and by RCB, for obtaining the BCD image that the 1401 produces with a binary card read.

Flow chart 9 shows subroutine RCK, for doing the reverse. This is used by RCB, to construct the binary card image for a BCD card (on the monitor input tape, of course, a card becomes BCD if it does not have 7-9 punches in column 1, regardless of how the simulated 1401 is going to read it later) which the 1401 is treating as binary; and by PCD, which punches decimal cards. Since the monitor punch-out tape must be entirely binary, the simulation of a decimal punch instruction has to be done by writing out the equivalent binary card image.

Flow chart 10 shows subroutine RCD, which carries out the 1401 operation of reading a card in the BCD mode, and storing the contents in 1401 cells 1-80, without disturbing word marks. There are two ERROR exits:

(a) if switch LASTC has been set on by a previous card-reading

operation, exit after printing TRIED*TO*READ*CARD*AFTER*PHYSICAL*
END-OF-DECK .

(b) if the record on the input tape is binary, the original card must have had 7-9 punches in column 1 (otherwise it would not have been accepted as binary by the monitor input program). Such a card would give a validity check if read by a 1401 in the decimal mode, and would halt the machine if the I/O check stop switch is on. The message for this is BCD READ OF BINARY CARD WITH I/O CKSTOP ON . If the switch is off, switch CDERR is set so that it can be tested by a 1401 "branch on card read check" instruction.

RCD is used in simulating the LOAD button to start a program (see flow chart 2) and by the specific routines for carrying out 1401 operations "read", "read and write", "read and punch", and "read write and punch"; "read and branch", "read write and branch", and so on. (See flow charts 22 and 24.)

Flow chart 11 shows subroutine RCB, which carries out the 1401 operation of reading a card in the binary mode, putting its binary image into cells 401-480 and 501-580, and putting its BCD image in cells 1-80, without disturbing word marks.
The possible error exit is the same as the first one described for RCD.

RCB is used by the specific routines for "read binary" and "read binary and branch". See flow chart 22.

Flow chart 12 shows subroutine PCD, which carries out the 1401 operation of punching a BCD card from cells 101-180; and subroutine PCB, which carries out the 1401 operation of punching a binary card from cells 401-480 and 501-580. Because the monitor punch-out tape is binary, PCD must begin by constructing the equivalent binary card image in a workspace.

The binary card image contains 160 characters of significant information; 8 characters with no significance follow them to bring the record up to its standard length of 28 7090 words.

PCD and PCB use the subroutine PCE, for which see immediately below. PCB is used by the specific routines for carrying out 1401 operations "punch binary" and "punch binary and branch". PCD is used by the specific routines for carrying out 1401 operations "punch", "read and punch", "write and punch", "read write and punch", "punch and branch", "read punch and branch", and so on. See flow charts 22 and 24.

Flow chart 13 shows subroutine PCE, which is used by PCD and PCB to decide, before punching out the current card, what pocket the most recently punched card went into, and to punch a "pocket change signal card" following it, if it did not go into the same pocket as the second-most-recently-punched card.

Three 7090 storage locations, KKLC, KKPR, and KPPR, are used to store the necessary information.
KKLC: "0" is stored in KKLC initially, and after every punching out of a card, to show that as far as is known so far, the card will go into the "normal" pocket, called "pocket 0" here.
(See the bottom of flow chart 12.) Whenever a 1401 K4 or K8 instruction is executed, "4" or "8" is put in KKLC (it is assumed that not more than one K4 or K8 is done between two punches) to show that the last card punched was to be sent to pocket 4 or 8.
Just before punching a card, if KKLC still contains "0", it is known definitely that the preceding card went to pocket 0; if KKLC

contains "4" or "8", it is known that the preceding card went to pocket 4 or 8. If "4" or "8", nothing more need be done by PCE, because the steps that will be described next have already been carried out during the execution of the K4 or K8 instruction (See flow chart 23.) But if KKLC contains "0", it is only now certain that the last card went to pocket 0; "0" is now stored in KKPR.

KKPR: This contains the number of the pocket into which the last-punched card went. However, "0" here is not definitely valid until just before the next card is punched (or the run ends), because a K4 or K8 instruction could intervene until then.

KPPR: Whenever "0", "4", or "8" in KKPR is known to be definitively valid, it is compared with KPPR. If they are the same, the last card punched went into the same pocket as the one before it. If they are not, a signal card with "N", "4", or "8" punched visibly on it is punched after the last card, to show that there was a change of pocket just before it. This is done by subroutine KPCH, which is mentioned at the bottom of flow chart 13.

Just after a card is punched (see the bottom of flow chart 12) the contents of KKPR are stored in KPPR, since what was the most recently punched card is now the second-most-recently punched card.

Flow chart 14 shows subroutines PRN and PWM, which are used to carry out the 1401 operations of printing the characters in cells 201-332, and printing the word marks in the same cells.
PRN is used by the specific routines for carrying out 1401 operations "write", "read and write", "write and punch", "read write and punch", "write and branch", "read write and branch", and so on.
PWM is used by the specific routines for carrying out 1401 operations "write word marks" and "write word marks and branch". (See flow charts 22 and 24.)

The difference between PRN and PWM is that PWM fills a work space with blanks and ones, corresponding to absence and presence of word marks in cells 201-331, and further processing uses the workspace instead of cells 201-331 as for PRN.

Both of them use subroutine NQD (see flow chart 16) to convert a series of 1401-type characters into a 7090 BCD record of 22 words, for eventual listing.

Since the monitor listing program uses the first character of each record to control line and page skipping, an extra character has to be written at the beginning of each 22-word record for listing. The character in 1401 cell 332 is dropped off to compensate for this, so it can never be printed by the simulator.

For simplicity, the leading character of the print record is added by saving the contents of 1401 cell 200 in location PHOL, putting the leading character in cell 200, and using subroutine NQD on cells 200-331. (The analogous procedure for PWM is obvious from the flow chart.) After the print record has been written, cell 200 is restored from PHOL.

The leading character is taken from location CARR, and then a blank is immediately put into CARR, as blank is the "neutral" leading character, which has no effect on line or page skipping. A non-blank will have got into CARR by the direct or indirect action of 1401 operation "F". (See flow chart 23.) F1 , F2, F3.....F@ will have put 1,2,3...or @ in CARR, and if this is transferred to the next print record as its first character, it will cause the monitor listing program to execute F1, F2, F3 etc. before printing the line.

After the print record is written out, location POSTP is examined. If it contains zero, i.e. a 1401 blank, nothing more need be done. If it contains 1, 2, 3,...or @, this was put there by a 1401 instruction FA, FB, FC,...or F), asking for a post-printing skip to channel 1, 2, 3,...or 12. So 1, 2, 3,...or @ is now put into CARR to provide that skip before the next line is printed, or the next non-printing skip is taken.

If POSTP contains / S or T, one, two or three blank print records are written. This was the result of a 1401 F/, FS or FT instruction, asking for a skip of one, two, or three lines after the next print line.

In any case, POSTP is zeroed before the exit from PRN or PWM.

PRN and PWM use subroutine WRUTE, shown in flow chart 15, to simulate imperfectly the use of channel 12 on the 1401 print carriage control tape. Normally, the tape has channel 12 punched just before the bottom of a page, so that before printing a line, the 1401 program may use a "branch on channel 12" instruction to see whether it is so near the end of the page that it should skip to the top of the next one (with an F1 instruction) before printing. Subroutine WRUTE maintains a count of how many lines have been written, normally, on the current page. If this count stands at 50, the simulator will allow a "branch on channel 12" instruction to branch; otherwise not. When the count reaches 51, it is returned to zero, as if a new page had bee begun. The count, stored in location WRUTA, is also set to zero whenever a 1, 2, 3... or @ is stored in CARR, whether by transfer from POSTP at the end of PRN or PWM, or directly by the execution of a 1401 instruction F1, F2, F3,... or F@. In such a case, a skip to a channel hole in the carriage control tape makes the existing line count meaningless. However, F1 is by far the most common, and is used normally to begin a new page, so the zeroing of WRUTA is appropriate.

Flow chart 16 shows subroutine NQD , which is used only by subroutines PRN and PWM to convert groups of 1401 characters into 7090 records for eventual printing. The flow chart should be self-explanatory.

The section of the simulator program from FFF to KKLC belongs to flow chart 23; see the explanation later on. FFF is a specific routine for the 1401 "control carriage" operation, and KKK for the "select stacker" instruction.

The section of the simulator program from RCRD to SEVA belongs to flow charts 22 and 24; see below for these. It contains specific routines for the 1401 instructions whose op codes are 1, 2, 3, 4, 5, 6, and 7.

Flow chart 17 shows the section of the simulator that begins when the 1401 has the address of the instruction it must now execute, and exits to INS1, INS2, INS4, INS5, INS7, or INS8 according to the length of the instruction.

If logical switch TRACE is on, the first thing that happens is the storage of the present I-address for eventual listing. Sub-routine DUMPA is used to convert the address to a blank and five decimal digits (see flow chart 49.) Sub-routine TRD (see flow chart 18) is used to write out a page of these addresses if it is full.

If logical switch SWICK is off, and the op-code of the 1401 instruction is B , N , V , or W , switch SWICL is set on; otherwise SWICL is set off. SWICL is tested by subroutines EVA and EVB (flow chart 19) when they find an invalid address to evaluate; if SWICL is on, they do not give an error halt, but evaluate the address as 8000.

The error messages are:

(a) CUT.OFF.BY.SWITCH.6 if the 7090 run is terminated with sense switch six (in order to regain control and get a dump if the 1401 program seems to be in a loop.)

(b) INSTRUCTION*ADDRESS*ABOVE*7999*OR*0 if the address in the I-address register is not between 1 and 7999.
(c) OP*CHARACTER*HAS*NO*WORD-MARK if the leftmost character of what is supposed to be the instruction has no word mark.
(d) INVALID*INSTRUCTION*LENGTH if what is supposed to be the instruction is 3 or six characters in length, or if the word beginning at the leftmost character of what is supposed to be the instruction is more than 8 characters long, and the op code is not slash or comma. Note that if the first character of the supposed instruction is B, and its fifth character is blank, the simulator does not look further to the right for a word mark, but accepts the instruction immediately as a 5-character instruction.

Just before INS in the simulator program, there are a number of storage locations that are frequently referred to, labelled LGTH, OP, ABIN, BBIN, ABINH, BBINH, CTRL and DCHAR:
LGTH contains PZE n, where n is the length in characters of the instruction being executed.
OP contains the op-code character of the instruction.
ABIN contains PZE MEM&n, where n is the contents of the A-address register of the simulated 1401. This number is correctly set immediately before and after an instruction is executed, but it is not altered progressively during the execution as it would be if it were the A-address register of a real 1401. This progressive counting is usually done in index register 1 of the 7090, which usually contains (-MEM-n). Therefore if the simulator detects an error during the execution of an instruction and ends the job with a dump, the dump will show the A-address register as

it was at the beginning of the instruction, and this may be a little different from what the A-address register of a real 1401 would contain at the moment of the corresponding halt.
BBIN is analogous to ABIN, but for the B-address register.
Index register 2 of the 7090 usually contains (-MEM-n), where n is the current B-address.
ABINH contains whatever ABIN contained just at the completion of the preceding instruction. This is needed for the "store A-address register" operation of the 1401.
BBINH is the same for BBIN and the "store B-address register" operation.
CTRL contains PZE MEM&n, where n is the 1401 address of the current instruction.
DCHAR contains the D-character of the current instruction, if it has one, or else the D-character of the last executed instruction that did have one.

Flow chart 18 shows subroutine TRD, which arranges 1000 addresses stored by the tracing function of the simulator into a page of 20 columns, to be read column by column, and outputs it for monitor listing. TRD is used by the simulator just after INS, if a page is full, and by the DUMP routine at the end, to output the last page of addresses at the end of the job.

The only reason why TRD is at all complicated is that the page is to be read by columns, but has to be written by rows. The first row must contain items 1,51,101,....951; the second must contain items 2,52,102,...952, and so on to the fiftieth row, which must contain items 50,100,150,...1000.

Flow chart 19 shows subroutines EVA and EVB, which evaluate MEM&a and MEM&b, where a and b are the binary equivalents of the A-address and B-address of an instruction, indexed if necessary. They use subroutine EV (see flow chart 20) to evaluate the unindexed A- and B-addresses, and the contents of an index register if necessary.

EVB is used only after control goes to INS7 or INS8, for a 7- or 8-character instruction. EVA is used twice in the simulator: just after INS7, for a 7- or 8-character instruction, and just after INS4, for a 4- or 5-character instruction.

There are two error messages:

(a) If the return from subroutine EV is on line 1, showing an impossible address in the instruction or in an index register, BAD*CHARACTER*IN*ADDRESS

(b) If the A- or B-address, before or after indexing, is above 7999: A*OR*B*ADDRESS*EXCEEDS*7999*OR*IS*ZERO

However, if switch SWICL is on, because switch SWICK is off and the address may not actually get used for memory access, there is no error exit, and the address is evaluated as 8000. This will cause an error exit if the address really is used for memory access.

Flow chart 20 shows subroutine EV, which evaluates the binary equivalent of a three-character 1401 address, ignoring the zone bits of the middle character. EV is used by subroutines

EVA and EVB (see above), and by the specific routine for the 1401 operation "modify address" (flow chart 41). The normal exit is return 2, but if one of the address characters is bad, return 1 takes place (which causes an error in flow chart 41, and in flow chart 19 if switch SWICL is off.)

Flow chart 21 shows the progress of the simulator between the time at which the length of an instruction has been determined (INS1, INS2, INS4, INS5, INS6, and INS8) and GO, where the op-code character is decoded and control is sent to the specific routine for the 1401 operation. The only subroutines used are EVA and EVB (see above).

Note that if the second character of a 5- or 8-character instruction is (, the A-address is not evaluated. A real 1401 would put a meaningless number in the A-address in such a case. Thus the final state of the simulator would differ from that of a real 1401 if a dump occurred immediately afterwards. If a 1401 program executed such an instruction and then executed an instruction that contained no A-address but used the contents of the A-address register, the results would be different for the simulator from what they would be for a real 1401. However, it is most unlikely to happen.

Just before subroutine EVB is called, the program turns off switch SWICL unless the op-code is N. Whereas an A-address may not be used for memory access in a conditional branch or a no-op, only in a no-op can a B-address not be used for memory access.

What happens at GO is described below flow chart 21. If the op-code of the instruction is an impossible one, the error message is NO*SUCH*OP. Following GO3, GO2, GO1, and GOZ in the simulator program one sees a TRA to each of the specific routines, with the op-code character to which it corresponds added as a comment. TRA ERRE is used for all the impossible op-code characters, to give the message NO*SUCH*OP.

The flow charts from number 22 on concern these specific routines.

Here is a table of all the valid op-code characters, with the symbolic address of the corresponding specific routine for each one, and the number of the flow chart where it is shown:

OP-CODE	ADDRESS	FLOW CHART	OP-CODE	ADDRESS	FLOW CHART
1	RCRD	22	N	INS	17
2	PRT	22	P	MVR	37
3	THR	22	Q	SAR	33
4	PCH	22	S	SUB	38
5	FIV	24	U	CTT	29
6	SIX	24	V	BWM	36
7	SEV	24	W	BBT	36
8	SRF	24	Y	MVZ	33
9	SPF	24	Z	MZR	34
A	ADD	38	=	MOD	41
B	BRA	43	@	MPY	44
C	CMP	40	Plus 0	ZAD	42
D	MVD	33	.	HLT	46
E	EDT	48	)	CWM	33
F	FFF	23	-0	ZSB	42
H	SBR	33	/	CLM	35
K	KKK	23	,	SWM	33
L	LOD	26	(	DIV	47
M	MOV	28			

Notice that for op-code N ("no operation") the simulator does not go to a specific routine, but branches straight back to INS to execute the next instruction, having done nothing but refill the I-, A-, and B-address registers.

For any op-code not found in the preceding table, there is an error exit with the message NO*SUCH*OP .

Flow chart 22 shows the specific routines for op-codes 1, 2, 3, and 4. They use sub-routines RCD (flow chart 10) , RCB (flow chart 11), PRN and PWM (flow chart 14), PCD and PCB (flow chart 12). Flow chart 24 shows the specific routines for op-codes 5, 6, and 7. They use sub-routines RCD, PCD, and PRN. From both flow charts there are error exits for invalid instruction length, with the message INVALID*INSTRUCTION*LENGTH. Since there is no op-code except N for which all the possible instruction lengths (1, 2, 4, 5, 7, and 8) are valid, error exits with this message occur in each of the specific routines; and they will not be mentioned hereafter. From flow chart 22 there are also error exits for invalid D-character, with the message INVALID*D*CHARACTER. For every specific routine in which D-characters are used, this type of error is possible, and so it will not be mentioned again in connection with the corresponding flow charts.

Note "CLME" on flow chart 22. Every executed branch in a simulated 1401 program will send control in the simulator to CLME.

In the flow charts for many specific routines, there are switches, represented by lozenges, in which the two exits are marked not plus and minus, but INS and CLME. Such a switch will have been preset somewhere in the specific routine to INS if the instruction is to be followed by the next sequential instruction, and to CLME if it is to be followed by a branch (which is necessarily to the A-address).

Flow chart 23 shows the specific routines for the "control carriage" (F) and "select stacker"(K) instructions.

An F instruction can have a D-character of four kinds:
(a) unzoned, between 1 and @ . In this case, the routine puts the character in location CARR, where it waits to become the control character of the next line to be written out for listing; when the output tape is eventually listed, this character will be inserted into an F instruction, to produce the same skip to a control-tape channel as the simulated F instruction would have given if executed on a real 1401 with the same carriage control tape. (See flow chart 14, and page 19 above.) But before putting the D-character in CARR, the simulator sees whether there is another D-character already there, waiting to be simulated. If so, a record of blanks, with the waiting character as control character, is written out before the new D-character is put in CARR.
On the zeroing of WRUTA, see the top of page 19 above.
(b) plus-zoned, between A and) . In this case, the routine puts the corresponding unzoned character in POSTP, where it will wait

to be used as a control character until after the next occasion on which a line is written out for printing. If there is already a character waiting in POSTP, it is overwritten. Presumably a real 1401 will disregard, e.g., an FB instruction if it is superseded by an FA before the next print instruction. See the second paragraph of page 19 above.
(c) minus-zoned -- J, K, or L. In this case, one, two, or three lines of blanks are written out for printing, and the count of lines per page in WRUTA is increased accordingly. This corresponds to a 1-,2-,or 3-line skip on a real 1401.
(d) zero-zoned -- /, S, or T. In this case, the character is put in POSTP (obliterating any character that may already be waiting there), to cause the equivalent of a 1-, 2-, or 3-line skip immediately after the next occasion on which a line is written for printing. See the third paragraph of page 19 above.

The only error exits are for impossible instruction length or D-character.

A K instruction can have a D-character of 1, 2, 4, or 8. K1 and K2 refer to read stacker selection, and as the program does not simulate this, a D-character of 1 or 2 causes the instruction to be treated as a no-operation, if two characters long, or an unconditional branch, if five characters long.

K4 and K8 refer to punch stacker selection. The simulation of either of these insures that at the end of the 7090 run, a message LOOK FOR POCKET CHANGE CARDS IN THE CARD OUTPUT will be output for listing, surrounded by asterisks to give it prominence. One may consider that the punched-card output of the run consists of a number of groups of cards, sent into various stackers.

For example, suppose a series of 26 cards, which we shall represent by small letters a to z, is punched into various stackers. In the following list, each letter is followed by N, 4, or 8, showing which stacker the card went into:

aN bN cN d4 e4 f4 g8 h8 i4 j8 k8 lN mN n4 o8 p8 q8 rN sN tN
uN vN wN xN y4 z8

Now the groups are a-c,d-f,g-h,i,j-k,l-m,n,o-q,r-x,y and z.
If the first group of cards in the output goes into the normal stacker, it will not be specially indicated. But apart from this, a special marker card will be inserted after the first card of every group. (Note well that the marker card will be after the first card of a group, not before it as one might expect.) Representing marker cards by (N), (4), and (8), the 26 cards in the above list would be interspersed with marker cards in the output as follows:

a b c d (4) e f g (8) h i (4) j (8) k l (N) m n (4) o (8) p q
r (N) s t u v w x y (4) z (8) .

A marker card has holes punched around its perimeter, and a large visible N 4 or 8 punched in the middle. It is punched by subroutine KPCH, for which see flow chart 13 and page 17 above. KPCH is entered from subroutine PCE (see flow chart 13 and pp 16-17 above), from specific routine KKK (which is now being described), and from ENDK (see flow chart 50 and its discussion later.)

"Flow chart 25" is only a note on subroutine BTD. This is entered with PZE n in the accumulator, where n is less than 16000, and it puts the equivalent 1401 address into cells BTDU, BTDV, and BTDW of the 7090 memory. For instance, if n=7999, the 1401 equivalent address is I9Z, and BTD will put PZE 9,,3 in BTDU, PZE 9 in BTDV, and PZE 9,,1 in BTDW.

The working of this subroutine is obvious from its listing in the 7090 program. It is entered from specific routines SAR, SBR, and MOD, for op-codes Q, H, and =.

Flow chart 26 shows the specific routine for the "load" operation, except when this involves magnetic tape. There is a branch to IOMOV&1 for "load" instructions that involve magnetic tape, after setting sense indicator number 1 to show that the tape will be read or written with word marks.

The routine uses "DOWNA" and "DOWNB", which are not really subroutines but macros, whose definition will be found at the beginning of the 7090 program listing.

Flow chart 27 shows how DOWNA works -- it reduces the effective A-address by 1, by increasing the contents of index register 1 by 1. If the contents of the index register are greater than -MEM-1 , the effective A-address is 0 or negative, and there is an error exit with message A*OR*B*ADDRESS*EXCEEDS* 7999*OR*IS*ZERO

DOWNB is exactly the same, except that it modifies index register 2, and the effective B-address.

Flow chart 28 shows the specific routine for the "move" operation, except when it involves magnetic tape. At IOMOV in this flow chart, there is a branch to IOMOV&1 (see flow chart 29) for "move" instructions that involve magnetic tape, after resetting sense indicator number 1 to show that the tape will be read or written without word marks. In the "move and binary decode" and "move and binary code" sections, note that the A- and B-address registers are tested only once for each pair of characters to be moved, to see if a zero or negative address will be involved. If this causes an error message (A*OR*B*ADDRESS*EXCEEDS*7999*OR*IS*ZERO) and dump, it may happen one character earlier than a real 1401 would have halted. IT IS POSSIBLE THAT THE SIMULATOR WILL GIVE AN ERROR WHEN THE REAL 1401 WOULD NOT HAVE HALTED, if an odd number of characters is being moved, and if moving one more character than was actually moved would have caused location 0 of the 1401 to be used. Since in practice the "move and binary decode" and "move and binary code" are always used for an even number of characters, the simulator neglects this possibility.

Flow chart 29 shows the specific routine for instructions with op-code U, and also the continuation at IOMOV&1 for "move" and "load" instructions that involve magnetic tape. There is an error exit (message NO.SUCH.I.O.UNIT.TYPE) if the third character of the instruction is not U or B. The same error message and exit are given after IOMOV&1 if the eighth character of the instruction is not R or W, and in routine CTT if the fifth character is not M, R, U, B, or E. IOMOV&1 sets sense indicator number S if the tape is addressed in binary mode, and resets it if in BCD; CTT treats the tape as if it were addressed in the BCD mode in any case.

Both routines use subroutine RWA (see flow chart 30) to find the 7090 tape address equivalent to the 1401 tape address. The 1401 address is taken from the fourth character of the instruction; if this is not a digit between 1 and 6, the error NO.SUCH.I.O.UNIT.TYPE occurs. If the 1401 tape number was not mentioned on the control card (see flow chart 1 and page 8 above), the job is ended with a dump and the error message 1401.TAPE. NUMBER.NOT.ON.CONTROL.CARD . Otherwise, the 7090 equivalent tape addresses is found from the table given in flow chart 1, and made binary if necessary.

Before a "write" instruction is carried out, it is necessary to convert the record as it stands in simulated 1401 memory into a 7090 record beginning at location THOLD. This is done by subroutine SQB. (See flow chart 32.) When this has been done, the simulator tests switch UDISC (see flow chart 1 and page 4 above) to see how to handle the difficulty created by the fact that a 7090 can read and write only records containing an integral multiple of six characters. (It is true that it can read a record whose length is not an integral multiple of six characters, but it does so as if the record ended with enough zeroes to fill out the last word, and there is no way for a 7090 program to find out whether zeros at the end of the last word came from the tape, or were supplied in this way.) If switch UDISC is on, tapes are not to be mounted or dismounted in connection with the run, and the simulator can use the tapes without regard to other programs. It lengthens every record which it writes by an extra word at the beginning; this word is PZE n, where n is between 1 and 6. n is the remainder after dividing the number of characters in the 1401 record by 6, or n is 6 if this remainder is 0; thus n is the number of valid characters in the last word of the 7090 record. In reading, the simulator treats the first 7090 word of a tape record as not forming part of what has to be converted to a 1401 record, but as containing PZE n, where n is the number of valid characters in the last 7090 word.

If switch UDISC is off, however, the reading and writing of tapes has to be as nearly as possible compatible with what a real 1401 would do. If a record to be written does not contain an integral multiple of six characters, it is lengthened in its 7090 form by from 1 to 5 blanks if BCD, or from 1 to 5 zeros if binary. In reading, there is no problem for the simulator; every record on tape contains, as far as the 7090 is concerned, an integral multiple of six characters. Of course the 1401 may read less than the full record on tape, if the memory space is terminated by a group mark with word mark; in that case the simulator may treat the record exactly as a real 1401 would have done.

The choice according to switch UDISC is made, in the simulator program, by making location RTPDN contain the command TCH RTPD if the switch is off, or IOCP RTPDB,,1 if it is on. RTPDB will in any case contain PZE n, where n is the number of valid characters in the last word of the record. This is for writing. For reading, substitute R for W in the above 7090 symbolic addresses. Then RTPDB will not, of course, contain the correct PZE n if the switch is off. The choice of command for WTPDN and RTPDN is made soon after BEGIA, near the beginning of the simulator program, if switch UDISC is set on during the reading of the control card.

After a "read" instruction has been partly carried out by a 7090 read, it is necessary to convert the record in 7090 form into a series of characters in the simulated 1401 memory. This is done by subroutine SPB, for which see flow chart 31.

The redundancy and end-of-file/end-of-reel switches, represented by 7090 cells TAPER and ENDRL, are set and reset according to the same logic as a real 1401 would use; with one exception. If a tape mark is read in the binary mode by a 1401, the tape check indicator is set on; the simulator does not do this. However, one does not normally test for redundancy after testing for end-of-file, and the indicator is automatically turned off before the next reading or writing of a tape by either a real 1401 or the simulator.

Besides subroutines RWA, SQB, and SPB, the routines on flow chart 29 use subroutine LVSYS. This is not flow-charted. It allows the simulator to make sure that data channels A and B are inactive, and then to disable the channel traps so that the simulator and the "IOEX" program (used by the simulator for input-output in all cases except the simulation of 1401 tape orders themselves) will not conflict. For reasons described in the 7090 manual in connection with channel trapping, it is not enough to execute TCOA * in order to be sure channel A is quiescent, if channel A is enabled for trapping. The program to which trapping sends control must store in a memory location some indication of whether it is using the channel or not. In fact, "IOEX" uses locations CHXAC, CHXAC&1, and so on for channels A, B and so on, storing a zero in a word whenever it has finished with the corresponding channel, and a non-zero whenever it starts to use the corresponding channel. As "IOEX" stands at present in the IBSYS system, the octal addresses of the locations for channels A and B, i.e. CHXAC and CHXAC&1 in the symbolic version of IOEX, are 01104 and 01105. If IOEX is altered so as to change these absolute addresses, then the two references in subroutine LVSYS of the simulator will have to be changed accordingly.

At ENSYS on flow chart 29, and in the simulator program listing, the simulator enables the channel traps from location 735 (octal) so that input-output will be through and under the control of IOEX until the next time a 1401 tape operation is simulated. This enablement may not actually be necessary, as it is probably done whenever a routine of IOCS is entered. At the moment, 735 is the location from which the traps are always enabled in IOEX; if this changes in future, the instruction at ENSYS in the simulator will have to be changed correspondingly, or eliminated if it is in fact unnecessary.

Flow chart 31 shows subroutine SPB, for converting 7090 records into records in the simulated 1401 memory. The calling sequence is explained at the top of the flow chart.

If sense indicator S is on, the record is in the binary mode, and if off, in the BCD mode. If binary, each 1401 character can be deduced from a 7090 character merely by spreading the zone bits apart from the numerical bits. But if BCD, it is necessary to use table RDTAB -- see page 12 above.

If sense indicator 1 is on, the record is to be put into 1401 memory as if by a "read/write with word marks" instruction (load); and if off, as if by a "read/write without word marks" instruction (move).

By "0-8-5" in the flow chart is meant the character that would be punched on a BCD card as zero, eight, and five punches. When reading a tape with word marks, it is possible that the last character of the record may be 0-8-5. Being the last character, it cannot indicate a word mark on the following character, and so it is treated as a normal character. This may or may not be what the real 1401 does.

Subroutine SPB uses subroutine SPBL, which is shown at the side of the same flow chart. It also uses macro UPB, which is diagrammed in the same flow chart. UPB increases the effective B-address by 1, by reducing the contents of 7090 index register 2 by 1. If the contents of this index register are less than -MEM-7999 , the effective B-address is above 7999, and there is an error exit with message A*OR*B*ADDRESS*EXCEEDS*7999*OR*IS*ZERO .

The macro UPA is the same as UPB, except that it uses 7090 index register 1 for the simulated 1401 A-address register.

Subroutine SPB is used only by the routine on flow chart 29.

Flow chart 32 shows subroutine SQB, for converting 1401 records into 7090 records. The calling sequence is explained at the top of the chart.

Sense indicators S and 1 are used as for subroutine SPB.

SQB uses subroutine SQWA, which is shown at the right of the same flow chart.

There is an error exit with message TRIED*TO*WRITE*ZERO-LENGTH*TAPE*RECORD if "B" in the calling sequence is the address of a group mark with word mark; i.e. if the B-address of a 1401 tape-write instruction is the address of a group mark with word mark.

Subroutine SQB is used only by the routine on flow chart 29.

Flow chart 33 shows the specific routines for op-codes
,) Q H D and Y

There are no peculiar error exits. However, for "store A-register" and "store B register", one may note that the error for A-address too low is not found by the simulator until after the storing has taken place, but before the A-address register has been altered. A real 1401, on the other hand, would adjust the A-address register step by step, and halt when it reached zero, leaving one or two characters unstored.

Flow chart 34 shows the specific routine for op-code Z . There are no special error exits or remarks. Flow chart 35 shows the specific routine for op-code / . There are no special error exits or remarks.

Flow chart 36 shows the specific routines for op-codes W and V. There are no special error exits or remarks. Flow chart 37 shows the specific routine for op-code P. There are no special error exits or remarks.

Flow chart 38 shows part of the specific routines for op-codes A and S. Except for the case of a 4-character subtract instruction (subtract a field from itself), control goes down the left side of the flow chart while some preliminaries are being carried out, and then there is a branch at the bottom to either ADTRU, for a true-add operation, or ADCMP, for a complement-add. ADTRU is in the middle of the top of the sam. flow chart, and ADCMP constitutes flow chart 39.

No subroutines from other flow charts are used on flow charts 38 and 39. At various places on these two flow charts, it is noted that there is an error exit if the digital part of a character is greater than 10. The message for this is ADD*OR*SUBTRACT*MEETS*NON-DIGIT .

In a true-add operation, the sign of the B-field is left unchanged; but after a complement-add, the rightmost digit of the B-field will always be either plus-zoned or minus-zoned, according to the sign of the result, and never unzoned or zero-zoned, even though it was unzoned or zero-zoned initially and is still positive.

In any character of the A- or B-field, blank numerical bits are treated like zero numerical bits (i.e. binary xx0000 like xx1010). In the result in the B-field, however, a zero in any position will be represented by zero, not blank, numerical bits.

Zone bits and overflow are handled as described on page 19 of the IBM System Operation Reference Manual for the 1401 and 1460, File Number 1401/1460-01, Form A24-3067-0.

Flow chart 40 shows the specific routine for op-code C. There are no special error exits or messages.

Flow chart 41 shows the specific routine for op-code = (modify address). This uses subroutines EV (see flow chart 20 and page 24 above) and BTD (see page 31 above). The only peculiarity is on the right hand side of the flow chart, where the A- and B-address registers are altered and tested for validity in a slightly different manner from that used by a real 1401. This makes no difference to the running of a program, but if there is an error halt here, the A- and B-addresses and the contents of the B-field will not be quite the same as when a real 1401 halted.

There is another error exit if either the A-field or the B-field contains an impossible character, as shown by return 1 from EV.

Flow chart 42 shows the specific routines for the "zero and add" and "zero and subtract" operations. The only peculiarity is that for a four-character instruction, the error exit for A-address reduced to zero will leave the A- and B-address registers containing different numbers from what they would contain at the corresponding halt on a real 1401.

Flow chart 43 shows the specific routine for op-code B. There are no special error exits or messages.

Note that a "branch on character equal" with K or Z as its D-character resets the end-of-tape or overflow indicator, respectively, whether it results in a branch or not.

A five-character branch instruction with 9 as its D-character is treated as a "no operation". However, a five-character branch with @ as its D-character (represented by "8-4" in a lozenge on the flow chart) is treated as a branch if location WRUTA contains 50; as explained above on page 20, the simulator attempts to simulate a printer with a carriage control tape that has a punch in channel 1 at the top of the page, and a punch in channel 12 fifty lines below it.

A five-character branch instruction with minus-zero, record mark, or (as its D-character is treated as a no-operation, because there is no simulation of punch errors, printer errors, or the kind of processing check detected by "branch on processing check".

A five-character branch instruction whose D-character is none of 9 @ K L minus-zero / S T U Z record mark (A B C D E F blank or plus-zero is treated as an error.

Flow chart 44 shows the specific routine for the "multiply" operation. For the sake of simplicity and speed, this is not carried out by the method a real 1401 would use. The multiplier and multiplicand are evaluated as binary numbers, by subroutine MPA (see flow chart 45); they are multiplied by a 7090 multiply instruction, and the product is converted to decimal form and stored in the simulated 1401 memory by subroutine MPB (see flow chart 45).

If K, the length of the product field, is greater than 21 characters, or if P, the length of the multiplicand, is greater than 10 characters, or if K-P, the length of the multiplier, is greater than 11 characters, this is arbitrarily considered to be an error; the message before the dump is PRODUCT*OVER*21*OR* FACTOR*OVER*10*DIGITS . If K-P is less than 2, this is a genuine error; the product field and the multiplier are too short; and the error message is PRODUCT*FIELD*TOO*SHORT .

The 7090 multiplication is done with positive factors. If the product is not less than the 35th power of 2, it cannot be contained in one 7090 word or register; it is necessary to divide it by the 10th power of 10 and use the remainder from this division to fill in the low-order ten digits of the 1401 product, and the quotient to fill in the remaining high-order digits.

The factors are limited to ten digits each because all ten-digit numbers are less than 2 to the 35th, while an eleven-digit number may not be. The product of two such factors cannot be as great as 10 to the 20th, so dividing a long product by 10 to the 10th must give a remainder and a quotient that are both less than 10 to the 10th, and no division overflow can result.

Flow chart 45 shows subroutines MPA and MPB, which are used by the specific routines for "multiply" and "divide" (see immediately above for multiplication, and see flow chart 47 below for division.)

MPA ignores the zone bits of the characters. If the numerical bits of a character are all zero, it is treated as a zero; if they add up to more than 10, there is an error exit.

MPB converts a binary 7090 number to an unsigned 1401 number. There are no error exits.

Flow chart 46 shows the specific routine for the "halt" operation. There are no special checks or error exits. If the control card contained an H, logical switch "HALTS" is on, and the 7090 prints an on-line message for the operator, showing the contents of the I-, A-, and B-address registers as decimal numbers. The program then halts. If the sign key on the 7090 console is off when the start button is pressed, the 1401 program obeys the halt instruction and continues as if the 1401 start button had been pressed. Otherwise, pressing the 7090 start button causes an exit like an error exit, with the message "1401 HALT".

Though it is not shown explicitly in the flow chart, the routine uses subroutine DUMPA (see flow chart 49 below) to convert the register contents into decimal form for printing.

Flow chart 47 shows the specific routine for the "divide" operation. This is not carried out by 1401 logic, but by a 7090 division. However, this makes no difference to the result as long as the divisor and dividend are not too long.

If K, the length of the divisor, is greater than 10 characters, there is an arbitrary error exit with the message DIVISOR*OVER*10*DIGITS. This is because a decimal number of more than 10 digits may not be less than 2 to the 35th, and so may be too large for a 7090 divisor.

The B-address of the instruction refers to the leftmost digit of the dividend. The check for a sufficient number of positions to the left of this, before a word mark is encountered, is not made yet. The next check after K is found verifies that all positions to the left of the character at B, up to and including the nearest one with a word mark, contain zeros. This check is not made by a real 1401, but if those positions do not contain zeros, the real 1401 would give a wrong answer. The message for this error exit is LEFT*END*OF*B*FIELD*NOT*ZERO .

The next part of the routine begins at the B-position and looks rightward until it finds a character with zoning. If this is not standard plus or minus zoning, i.e. is zero zoning, there is an error exit with the message DIVIDEND UNSIGNED OR IMPROPERLY SO. If a character with a word mark is found before a signed character, the same error exit and message occur. Let P be the address of the signed character. Then P-B&1 is the length of the dividend. If this is greater than 20 characters, an arbitrary error exit occurs, with the message DIVIDEND*OVER*20*DIGITS. The reason for this is that a longer dividend might not be small enough for a 7090 dividend and even if small enough might produce a 7090 divide overflow with a 10-digit divisor.

There are no further error exits from this routine. If the divisor has been found to be zero, division does not take place, and the same procedure is followed as for a real 1401.

The binary equivalent of the divisor is found by subroutine MPA (see flow chart 45 and page 43 above). If it is longer than 10 digits, its high-order part is evaluated and multiplied by 10 to the 10th, and added in double-precision addition to the binary equivalent of the low-order 10 digits. As this sum stands in the AC, containing "x", and the MQ, containing "y", it is equal to x times 2 to the 35th, plus y. If x is not less than the binary form of the divisor, division cannot take place immediately. The divisor is repeatedly multiplied by 10, until a number greater than x is obtained. The dividend is divided by this number; the quotient Q1 and remainder R1 are saved. R1 is now divided by the original dividend, giving a quotient Q2 and remainder R2. R2 is the binary form of the final remainder. The final quotient is Q2 plus (Q1 multiplied by 10 as many times as the divisor had to be multiplied by 10). If the number of multiplications by 10 is "z", it is known that Q2 is smaller than 10 to the z-th. (R1 was smaller than the divisor times 10 to the z-th, so R1 divided by the divisor must give a quotient, Q2, smaller than 10 to the z-th.) So the program can combine Q1 and Q2 in the 1401 memory by putting the decimal equivalent of Q2 in the low-order z positions of the quotient field, and the decimal equivalent of Q1 in the remaining high-order positions.

If the length of the dividend is less than 10 digits, or if the high-order half of a longer dividend, when waiting in the AC and MQ to be divided, is greater than the divisor, the dividend can be divided immediately by the divisor, and the quotient and remainder are each converted to decimal numbers simply, and stored in the proper fields in the 1401 memory.

Flow chart 48 shows the specific routine for the "edit" instruction. The only special error exit occurs if the character at the A-address has a word mark; the message for this is TRIED*TO*EDIT*A*1-CHARACTER*FIELD . This routine uses no sub-routines.

The editing is done as if by a 1401 without the "expanded print edit" feature.

Flow chart 49 shows the subroutine DUMPA, which converts a 7090 address into a blank and five decimal digits that give the 1401 equivalent of the 7090 address. "MEM", the 7090 address of location 0 in the simulated 1401 memory, is first subtracted from the number, and the remainder is converted to decimal.

DUMPA is used in flow chart 17 (see page 21 above) to make printable addresses when tracing has been requested on the control card; by the specific routine for the "halt" operation (see flow chart 46 and page 44 above) to get printable addresses for an operator message; and by the routine DUMP (see flow chart 50 immediately below) to get printable addresses for listing in the final dump.

Flow chart 50 begins at ERROR, which is not actually the name of an instruction in the simulator program, but represents the various points in the program at which error messages are printed, and from which the program always branches to DUMP. (See the definition of the macro ERRR at the beginning of the simulator program listing.)

Beginning at DUMP, the following things are done:
(1) If the control card contained an asterisk, so that logical switch "TRACE" is on, and if a partially-filled page of addresses is waiting to be written out, subroutine TRD (see flow chart 18 and page 23 above) writes it out.
(2) The contents of the I-, A-, and B-address registers are written out for listing, along with "OFLO", "TPCK", "ENDRL", "A=B", "UNEQ", "AHIGH", and "BHIGH". After each of these words, either "ON" or "OFF" is written, according to whether each of the indicators for overflow, tape redundancy, end-of-file or end-of-tape, A = B, A not = B, A greater than B, or A less than B is on or off.

(3) A dump of the 1401 memory is written out for listing.

(4) If location KCARD contains non-zero, at least one K4 or K8 instruction has been simulated, and the punched-card output will contain one or more pocket-change cards (see flow chart 23 and page 29 above). As a warning,

```
***************************************************
* LOOK FOR POCKET CHANGE CARDS IN THE CARD OUTPUT *
***************************************************
```

is written immediately after the dump.

In addition, if the last card punched was sent by the program to a different pocket from the one preceding it (or if only one card was punched and it went to pocket 4 or 8) the appropriate pocket change card is written out for punching.

(5) If the control card contained a dollar sign, so that logical switch CDUMP is set, a card with MEMORY visibly punched on it is written out for punching, followed by 100 binary cards that reproduce the contents of the 1401 memory. (See pages 4 ff. above.)

(6) CALL EXIT is executed, terminating the 7090 job.

HOW TO USE THE 1401 SIMULATOR

1. Get a copy of the small binary program deck (about 20 cards). Substitute your ID card for the sample ID card near the beginning of this deck. If the 1401 program may punch out some cards, put a "Punch Binary Cards" control card immediately before the ID card.

2. After the last card of the program deck, put a 7-8 card. Then put a control card for the simulator. This card will contain various characters, punched continuously from column 1. The program will stop scanning the control card when it encounters the first blank column, so any characters punched to the right of the first blank will be ignored. I.e., punch the control characters continuously, with no blanks interrupting them.

The control characters and their meanings are as follows:

& (12-punch) The simulated 1401 will behave as if the I/O check stop switch were on. If this character does not appear on the control card, it will behave as though the switch were off.

* Every time the simulated 1401 is about to execute an instruction, the address of that instruction will be written out, to help in tracing errors. The address will be in the form of a five-digit number; the numbers will be arranged in columns of fifty, twenty columns to a page. Such pages are to be read by columns, not by rows. They will be mixed up in the output with whatever pages the simulated 1401 has printed, but this should cause no confusion.

If the control card does not contain an asterisk, this tracing will not be done.

A B C D E F or G The simulated 1401 will behave as though every sense switch whose letter appears on the control card were on, and every sense switch whose letter does not appear on the control card were off.

H Whenever a programmed halt occurs in the simulated 1401 program, the 7090 will halt, and print a message on-line, showing the contents of the I-address register, A-address register, and B-address register, and telling the operator:
PROGRAMMED 1401 HALT. PRESS START TO CONTINUE. TURN ON SIGN KEY AND START TO END 1401 JOB.
The 7090 operator must have been instructed to check the contents of the 1401 registers as printed, and continue or terminate the 1401 job accordingly, by having the sign key on the console off or on, respectively, when the start button is pressed.

If there is no H in the control card, the first programmed halt in the 1401 program will terminate the job automatically.

K Whenever an instruction with an invalid A- or B-address is executed, the job will be terminated as for a 1401 error halt. If there is no K in the control card, invalid addresses in "no-op" instructions will be tolerated, and so will invalid A-addresses in conditional branch instructions if the condition is not met and the branch does not take place. This is more nearly the way the 8-K 1401 at CETIS works at present.

1H 2H 3H 4H 5H 6H For every tape which the 1401 program will use,
1L 2L 3L 4L 5L 6L the number (i.e. the 1401 address number) and density must be given as two characters on the control card. If the 1401 program addresses any tape that has not been mentioned on the control card, the job will be terminated automatically.

1401 tape number	1	corresponds to	7090 tape	A-5.
" " "	2	"	" " "	B-5.
" " "	3	"	" " "	A-6.
" " "	4	"	" " "	B-6.
" " "	5	"	" " "	A-7.
" " "	6	"	" " "	B-7.

Just as one would instruct the 1401 operator, in terms of 1401 tape numbers, what tapes to mount on what units, so one must instruct the 7090 operator, in terms of the corresponding 7090 unit numbers, what specific tapes and work tapes to mount before the job and remove afterwards.

If the control card contains any 1401 tape number(s), the simulator program will halt when it has finished reading the control card. In this case, it is essential to instruct the operator, on the job card, either (a) to make sure the proper tapes have been mounted, and then press START, or (b) simply to press START if only work tapes will be used by the 1401 program. (It is possible to get messages printed on-line by the 7090, which the operator will look at during this halt. This involves putting comment cards somewhere near the ID card in the input deck; it is left to the reader to investigate this detail of the IBJOB system.)

If the control card does not contain any tape numbers, the simulator does not halt automatically at this point.

T The 1401 program will be initiated by simulating the pressing of the LOAD TAPE button, i.e. by the sequence of instructions
/080 ,001 M(U1001R B001

If the letter T does not appear in the control card, the program will be initiated by simulating the pressing of the LOAD button on the card unit; i.e. /080 ,001 1001

$ At the end of the job, the contents of the 1401 memory will be punched out on 100 binary cards (or rather, written on tape for later off-line punching.) These 100 cards will follow all the cards that may have been punched out by simulated 1401 instructions, and they will be preceded by a marker card with MEMORY punched visibly on it. The MEMORY card should be discarded, and the 100 cards that follow it preserved for possible use as explained in the next paragraph. If $ does not appear in the control card, these 101 cards will not be punched.

= If the equal sign appears on the control card, the control card must be followed immediately by 100 cards that were punched after a previous job, as explained in the preceding paragraph. These cause the 1401 memory to be returned to the state it was in at the end of that previous job. The next card after those 100 cards must have either blanks or a 4-digit decimal number punched in columns 1-4. This is a "transfer" card. The next card after the transfer card can be thought of as being the first card in the deck placed in the card reader of the simulated 1401. If the transfer card is blank, the program is initiated by simulating the pressing of the LOAD TAPE button or the LOAD button on the card reader, depending on whether the control card contained a T. Otherwise, the four digits at the beginning of the transfer card are taken to be a 1401 address, and the program is initiated by branching to that address.

If the equal sign does not appear on the control card, the 1401 memory initially contains random information. The first card after the control card can be thought of as the first card in the deck placed in the card reader of the simulated 1401. The program is initiated as described in paragraph T above.

U If this letter appears on the control card, the simulated card reader will not take the cards that follow the control card as the input deck. Instead, it will assume that the immediately preceding job in the monitor was a compilation which left a copy of the program(s) and data in the "common" section of disc storage. Therefore, the simulator will take what is in that part of the disc storage as the input deck. See "How to Load the Simulator Immediately after the COmpiler", at the end of these notes.

If the control card does not contain "U", the simulation of card reading takes place in the ordinary way, as described in the various notes on the simulator program.

3. After the control card, if it contains no = , or after the 100 cards that must follow it if it does contain = , put whatever cards would be loaded in the card reader of a real 1401. If it happens that the program does not call on the card reader, or if the control card contains "U", put a few blank cards there anyway -- say 10.

Before the last card of the deck that would be loaded in a real 1401, insert a card with:

```
/////THE NEXT CARD WOULD HAVE BEEN PHYSICALLY LAST IN THE 1401 INPUT.
```

punched in columns 1-70, and blanks in columns 71-72.

This artificial indication of the end of the input deck is needed because a tape mark on the monitor input tape does not represent the end of the deck; it represents a 7-8 card. Whenever the card input to the simulated 1401 includes a card with two holes in column 1, in the 7- and 8-rows, this will become, on the input tape for the 7090 monitor, a tape mark. When the simulated 1401 meets a tape mark in its "card" input, it interprets it as a card with the 7-8 punch in column 1, and blanks in columns 2-80. If a card has a tape mark in column 1 and information in columns 2-80, the simulated 1401 is unable to read the information.

4. After the last card of the 1401 input, place a 7-8 card, $IBSYS and $EXECUTE FORTRAN control cards, and another 7-8 card.

5. On the job card, you may instruct the operator that if the time limit is exceeded, he should set sense switch 6 of the 7090 to on; this should cause the simulator to terminate the job.

6. Apart from this use of sense switch 6, and of key S at a programmed 1401 halt, the sense switches and the 36 input keys on the 7090 console are ignored by the simulator.

7. Whenever a 1401 job is terminated, whether through a programmed halt or through an error halt, the simulator writes an appropriate message on the monitor listing tape, then the contents of the simulated I-address, A-address and B-address registers, and the settings of all the simulated indicators, and finally a complete dump of the simulated 1401 memory.

8. Apart from input-output, multiplication, and division, the simulator is intended to behave just like the 1401-2 at CETIS. Here are the essential remarks about the simulation of these functions:

CARD READING: Any cards that must be read by the simulated 1401 will have to be read by the monitor card-to-tape program. This is naturally beyond the control of the simulation program, and it means that only two types of cards can be accepted -- those in which all 80 columns contain something that would be valid for BCD reading (i.e. not more than one numerical punch, or 8 with one other numerical punch; not more than one zone punch, but 0 may count as numerical in the combinations &0 and -0); and those that have rows 7 and 9 punched in column 1, but not rows 4, 5, 6, and 8 in that column.

The former type of card will be converted to a BCD record by the card-to-tape program, unless column 1 contains the 7-8 punch for a tape mark. Then the card will be converted to a tape mark on the tape, regardless of the contents of columns 2-80.

The latter type of card will be converted to a binary record.

A BCD card image on the input tape can be read by a simulated 1401 instruction for either binary or BCD reading. But a binary card image represents a card with 7-9 in column 1, which would certainly cause a read check if a 1401 attempted to read it with a BCD instruction. Therefore a simulated BCD read instruction, applied to a binary record on the input tape, will cause a read check but will put into the 1401 memory just what a real 1401 would have read from the original card, if it tried to read it in BCD.

This is the only occasion on which a read check can occur for the simulator. If the control card contained a plus sign, the I/O check stop switch is simulated on, and there is a dump and termination of the job, representing an error halt on a real 1401. Otherwise, the card read check indicator can be tested and turned off by the appropriate 1401 instruction.

The real 1401 has the instructions K1 and K2 for directing cards into special pockets after reading them. No attempt to simulate this is made, and such instructions are treated as no-operations or as unconditional branches, depending on whether they are 2-character or 5-character instructions.

CARD PUNCHING: As already pointed out, if the simulated program is going to punch any cards, it is necessary to insert a "PUNCH BINARY CARDS" control card just before the ID card in the input to the monitor.

The 1401 program is free to punch BCD or binary cards. They will come back to the programmer, in the monitor output, with the usual extra cards before and after -- at least an ID card before and a 7-8 card after.

The real 1401 has the instructions K4 and K8 for directing cards into special pockets after punching them. If any such instructions are executed during a job, a message will be printed at the end of the monitor output:

```
****************************************************
* LOOK FOR POCKET CHANGE CARDS IN THE CARD OUTPUT *
****************************************************
```

When the output cards are obtained, one should do the following:
(a) look through the output deck for any "pocket change cards" (if the message quoted above was not printed, there will not be any). These cards have a double row of holes punched around all four edges, and a visible N , 4 , or 8 punched in the center. Every pocket change card should be moved forward one place in the deck; i.e., it should exchange places with the card that immediately precedes it.
(b) if the first card in the deck is not now a pocket change card, put a pocket change card with N in the center at the beginning of the deck.
(c) deal the cards into three piles, an N pile, a 4 pile, and an 8 pile. Whenever a pocket change card is reached, it should be thrown away, and all the cards following it should be dealt onto the pile it named, until the next pocket change card is reached.

One will then have, in the N pile, all the cards that a real 1401 would have punched into the normal pocket; in the 4 pile, those that a real 1401 would have punched into the 4 pocket; and in the 8 pile, those that a real 1401 would have punched into the 2/8 pocket.

Obviously, steps (a), (b), and (c) above can be combined in a single handling of the output deck, but the explanation is clearer if the steps are separated.

No punch checks can occur.

PRINTING: The simulation program will behave like a 1401 in which the rightmost position on the printer always printed blank, whatever character 332 in the memory might be.

Whenever a real 1401 would print a record mark, the simulator will produce a dollar sign in the same position. This is because the output from the simulator has to be listed by the ordinary monitor listing program, which treats every record mark as an end-of-line signal instead of printing it.

The simulator can simulate F-instructions by writing out records with the appropriate control characters at the beginning, which the listing program will translate into F-skips afterwards. However, the output has to be listed by the ordinary monitor listing program, so only channels 1 and 12 of the printer carriage control tape will be significant.

A real 1401 can use a branch instruction to test whether the printer carriage is at a point corresponding to a punch in channel 12 of its control tape. The simulator maintains a counter which is initially zero, and is increased by one every time a line is printed. When it reaches 50, the simulator behaves as though a punch in channel 12 of the carriage control tape were available for sensing. When it reaches 51, it is immediately reset to 1. When an F-instruction causes one, two or three lines to be skipped, the same number is added to the counter, with a return to 1, 2, or 3 if 51, 52, or 53 is reached. When an F-instruction that causes a skip to some channel punched in the control tape is executed, the counter is set to 1.

MAGNETIC TAPE READING: If a tape record being read contains an integral multiple of six characters, there is no difficulty. If it does not, the 7090 will read it as if it contained an extra 1 to 5 zeros, to make up a total which is a multiple of six.

If the 1401 program happens to provide a group mark with a word mark in the memory, correctly defining the length of the record, any extra zeros of this kind at the end of the record will be ignored by the simulator.

MAGNETIC TAPE WRITING: If the number of characters in a record to be written is not an integral multiple of six, the simulator has to extend it with from 1 to 5 extra characters, before the 7090 can deal with it. The extra characters will be blanks if the record is BCD, and zeros if binary.

MULTIPLICATION AND DIVISION: Limitations are put on multiplication and division, to avoid dividends and products that will not fit in the AC-MQ of the 7090.

In multiplication, the product field must not be more than 21 characters long, the multiplicand must not be more than 10 characters long, and the multiplier must not be more than 11 characters long.

In division, the dividend must not be more than 20 characters long, and the divisor must not be more than 10 characters long.

There are a few conditions in which a real 1401 would produce a wrong answer for a multiplication or division, because the program had not set up the fields correctly, but in which the simulator will make an error halt and dump.

HOW TO LOAD THE 1401 SIMULATOR FROM SCRATCH

The small binary program deck depends on finding most of the the simulator program on the discs, in cylinder 248 of module 0. If the program is to be modified, or if something has happened to the information on the discs, it is necessary to begin again with the complete binary program deck. One should load the following into the monitor input:

(1) $IBSYS
(2) $RESTORE
(3) $EXECUTE   IBJOB
(4) your ID card
(5) $IBJOB   NOSOURCE,MAP,FILES
(6) the binary program deck for AFRB, from the $IBLDR card, AFRB0000 , to the $DKEND AFRB card, which at present is AFRB0235 .
(7) a 7-8 card
(8) a card with A in column 1, and blank otherwise. This will be the simulator control card.
(9) a card with ,008009. in columns 1-8 . This is a minimal "program" for the simulator to execute.
(10) a 7-8 card.
(11) $IBSYS
(12) $EXECUTE   FORTRAN
(13) a 7-8 card.

The output from this monitor job will naturally be completely uninteresting. Items (8) and (9) above could be replaced by a control card and 1401 program that did have some interest, but we are assuming for the moment that what is wanted is not to do any particular job of 1401 simulation, but merely to get the simulator program established in disc storage.

Once this has been done, one can prepare the small binary deck as follows:

(a) From the binary program deck for AFRB, copy all the cards beginning at AFRB0000 and ending with the card that contains location CHARGE; at present this is location 00063 (octal) in the program, and is to be found on card AFRB0009.

If some future modification of the simulator program altered the location of CHARGE, or the number of the program card that contained it, one would have to take account of this in preparing the small binary deck. But for the remainder of this description we shall assume the values 00063 and AFRB0009 without further comment.

(b) Prepare a card which is to replace everything after AFRB0009 in the complete binary deck, up to but not including the $CDICT AFRB card. This card must contain the equivalent of

BSS n	where n is the number of locations occupied by the rest of the complete binary deck.
END CHARGE	

To find n , look for the page headed "AFRB CONTROL DICTIONARY" near the end of the MAP assembly listing. This page begins with the listing of a BCD card containing "$CDICT AFRB". Then follows the listing of a binary card; one sees on the page something like this:

```
BINARY CARD ID.  AFRB0233
             036056000062                          PREFACE  etc. etc.
```

The total length of the program is contained in the decrement of the first word of information on this card; the length in the example above, which is taken from the present state of the program, is octal 36056 . This will change slightly whenever the program is re-assembled, but let us accept it as 36056 for the rest of this discussion.

From this number, 36056, subtract the octal address of the first word of program on the next card after the one containing location CHARGE; i.e. on the first binary program card that is not being copied in the short deck. Since CHARGE falls at present in card AFRB0009, we look at the MAP assembly listing to find that the octal address of the first program word on card AFRB0010 is 00066. Subtracting 00066 from 36056 gives octal 35770.

So we need a binary program card containing the equivalent of:

```
    BSS  n    where n is the decimal equivalent of octal 35770
    END CHARGE
```

and we look up CHARGE in the assembly listing and find that its octal equivalent is 00063 .

Therefore we punch a column binary card with the following in its first seven words (octal representation):

```
    704502000003
    000000000000
    013600000000
    000000000000
    000000000000
    200000035770
    000000000063
```

The only variable parts of this card are the 00003 at the end of the first word, the 35770 at the end of the sixth word, and the 00063 at the end of the seventh word. The derivation of the 35770 has just been explained at length above, and the 00063 is the equivalent of CHARGE. The 00003 is one more than the number in the same position of the preceding card. After the $TEXT AFRB card, succeeding cards in the deck contain consecutive numbers in this position, beginning at 00000 for the card next after $TEXT AFRB . The last card we retained, AFRB0009 as things stand, has at present 00002 in this position because it is the third card after $TEXT AFRB .

In columns 73-80 of this card, punch AFRB0010, assuming that AFRB0009 is the number of the last card retained from the complete deck.

Now take the last part of the complete program deck, beginning with the " $CDICT AFRB " card, and copy it; but modify the plugboard of the reproducer so that columns 77-80 of the new cards are left blank. Into these columns, insert serial numbers following on from the number of the last card prepared (as this was AFRB0010 in the current example, these numbers will be 0011 ff.)

One more modification has to be made to the first card following the " $CDICT AFRB " card (number AFRB0012 in the current example). The address part of the third column binary word on this card gives the address at which the program will begin to be executed. As the program is assembled, this address is LOAD, octal 00062; but in the small binary deck it must be CHARGE, octal 00063. As long as LOAD and CHARGE have these octal equivalents, the modification of the card is very simple: put an extra punch in the 9-row of column 9, altering 00062 to 00063, and an extra punch in the 0-row of column 1, showing that the check sum is no longer valid.

There is no reason to expect that 00062 and 00063 will change through re-assembly of the program, because what lies between locations 00000 and 00063 in the program is concerned only with copying the program itself to and from the discs, and there is no inducement foreseeable for changing this. But if these addresses should change, the final card modification described in the previous paragraph would become more complicated. Presumably the change would affect only the last four octal digits of the address; the simplest thing then would be to copy the card once more, with the plugboard of the reproducer altered to leave column 9 of the new card blank. Then one would punch into column 9 the equivalent of the last four digits of the octal equivalent of CHARGE. It would still be necessary to add the punch in the 0-row of column 1, to show that the check sum had been invalidated.

(c) At the beginning of this shortened binary deck, put the first five of the control cards mentioned at the top of page 8 above; namely, $IBSYS, $RESTORE,$EXECUTE IBJOB, your ID card, and $IBJOB NOSOURCE,MAP,FILES .

We now have a new small binary program deck, which can be used as described at the beginning of "How to Use the 1401 Simulator". This procedure must be gone through whenever the simulator program is re-assembled.

Minor corrections to the simulator program can be made with patch cards. Their format is explained at the beginning of the commentary on the flow charts for this simulator. Near the end of the MAP assembly listing of the simulator, there is a card: PATCH BSS 200 which reserves 200 cells for putting program corrections in without disturbing the total length of the program. (Typically, a program correction, if done by patching, requires the replacement of an instruction in the program by a branch to some point outside the program, where the new version of whatever is being corrected has been loaded. The space reserved inside the program by the PATCH card is for such "new versions".)

Patch cards, if any, are to be put immediately after the 7-8 card that follows the small binary deck, and before the simulator control card. The simulator program will note that there are patch cards, and after the last one has been read and obeyed, it will copy itself in its newly-modified form onto the discs.

(This is different from the system in the compiler program described elsewhere. If that program is to be modified by patch cards, it has to be loaded from the complete binary deck, as if it were not already on the discs.)

When the simulator is re-assembled by the MAP assembler, it is possible to use it immediately afterwards, during the same monitor. But this does not necessarily put the program correctly on the discs. It must still be loaded at least once from the complete binary program deck.

HOW TO LOAD THE SIMULATOR IMMEDIATELY AFTER THE COMPILER

The simulator program can be loaded in the monitor with " U " punched on the control card, in addition to any other necessary characters, and with no program cards following the control card.

This is only to be done if the simulator program immediately follows a compiler job in the monitor input. The program or programs that the compiler compiles will then be executed in the same order by the simulator.

There is an important limitation on this. No magnetic tapes are to be mounted before the programs are executed, or dismounted afterwards. The way in which the simulator will read and write tapes, during such execution, is incompatible with normal 1401 operation. Since there is no need for the 7090 operator to do anything about tapes, the simulator program will not halt after reading the control card, even if it names one or more tapes.

So if one wants to compile and test immediately, on the 7090, a 1401 program that -- say -- reads a magnetic tape, does something with the information, and writes it on another tape, it will be necessary to give the following as input to the compiler:

(a) a compiler-language program for reading in information on cards and copying it onto tape in a suitable format for use as input by the main program.
(b) the cards for (a) to use.
(c) a " (PROG) " card, followed by the main program in compiler language.
(d) a " (PROG) " card, followed by a compiler-language program for reading from tape the output of the main program, and printing it.

Sometimes this will be more trouble than it is worth, but often items (a) and (d) will be perfectly simple, and the testing immediately upon compilation will save waiting half a day to find out, when a 1401 becomes available, that there is an error in the main program that was compiled.

To disseminate knowledge is to disseminate prosperity — I mean general prosperity and not individual riches — and with prosperity disappears the greater part of the evil which is our heritage from darker times.

Alfred Nobel