July 15, 1963

NEWSLETTER

Committee on Computers in Research
Federal Reserve System

Programmers, economists, and others in the Federal Reserve System are invited to submit contributions to the Newsletter. Contributions may consist of program routines, programming techniques, computer applications for economic and statistical research, and similar matters of interest to System personnel.

Contributions may be submitted to members of the Committee on Computers in Research or to Emanuel Melichar, Economist, Division of Research and Statistics, Board of Governors.

CONTENTS

NEWS NOTES

Richmond

We have now programmed the majority of current reporting series which were previously tabulated on conventional equipment. These included the F.R. 413, F.R. 422, and F.R. 635 reports, the daily tab of reports of deposits subject to reserve requirements, the departmental report, quarterly interest rate survey, and the call report.

In addition, the consumer credit report for commercial banks and phases of the department store sales and stocks reports have been computerized. The consumer credit program is being revised to include minor editing changes and a listing for the Board's preliminary report.

In the immediate future our major efforts will be concentrated on the electric power series, the F.R. 416 report and operating ratios. (The 1962 operating ratios slip sheets, which included a cut-off verification section, were printed by the computer.) We also plan to use the computer extensively in converting the debits series to the new format and to write a program for the computation of an exponential curve.

Elizabeth W. Angle

San Francisco

We have been testing a number of time series for the efficiency of the twelve-month moving average as a means of representing the trend-cycle component. The technique used to make the test is that included in Milton Moss' paper prepared for the Seminar on Seasonal Adjustment. We have modified a Census II program to incorporate the adjustments indicated by the Moss method. In some series the change in the seasonal factors has been substantial after the modification. We have also found in a couple of series that the single modification was not adequate.

Donald Snodgrass

New York

We have written a two-part program in 1410 ForTran that "solves" the reduced form equation of a modified version of Lawrence Klein's quarterly econometric model of the postwar U.S. economy.

The output of the program consists of five tables, each showing the changes in GNP and its components in the first four quarters and in the full year following initiation of an exogenous shock to the economy. The first table shows the results when the exogenous shock

is a lump-sum increase in personal income taxes; the second, when the shock is an increase in allowable exemptions; the third, when the shock is an increase in the marginal tax rate for individuals; the fourth, when the shock is an increase in government spending; and the fifth when the shock is an increase in expenditures for new plant and equipment.

The program requires as inputs one parameter card containing the values of the slope coefficient of the personal tax equation and of the slope and intercept coefficients of the corporate tax equation and four data cards, each of which contains the values of 14 predetermined variables in the model--one card for each of the four quarters following the initiation of the exogenous shock.

Ralph C. Schindler

Atlanta

We have completed the programming of the Annual Report for Department Store Sales and Stocks, by Departments. The entire report is produced, using monthly summary cards. The printed output which includes all labels and titles is reduced and used for final copies.

"Cotton-picker"--prints and adds to totals up to six different fields from a tape record for one breakdown. We are in the process of modifying the program to take care of two or more breakdowns.

Jack Seifert of Methods and Systems Department has modified the Philadelphia Simple Correlation program to punch cards which can then be used as input for a plotting program.

We are writing several programs to produce a file of back data for Consumer Credit to be used for permanent records.

Grace M. Kisner

Boston

Our recent new activity has been to compute the historical series for a new Production Index for New England. This was done on the IBM 1620 computer.

Howard Smith

Philadelphia

The X-9 version of the Census Method II seasonal adjustment program for the 1401 has been completed, tested and made available to other Reserve Banks and to the Census Bureau.

The program was tested both by hand calculations and by comparisons with results produced on the Board's 1410 and the Census

Bureau's Univac versions. Some differences occur between these three results on some series, as was true of the previous Method II programs. After spending a day at Census reviewing the differences and discussing methods, it was agreed that the differences are insignificant and are due to slight differences in programming techniques.

The program in its present form was written for the minimum configuration 4-K 1401 computer and requires somewhat more card handling than our Census Method II, X3 program. Later it will be modified to operate on 8-K equipment without intermediate card handling.

It also may be newsworthy that an additional 1401 computer has been installed here primarily for use of the Research Department. It is an 8-K card system with the following special features:

Advanced programming
Multiply-divide
High-low-equal compare
Sense switches
Print control, additional.

K.M. Snader

FORTRAN NOTES

1410 ForTran Improved

The new version of 1410 ForTran is a tremendous improvement over the previous one. By eliminating the autocoder phase compile time is reduced to less than half that formerly required. Several programs have been compiled with the new version and no bugs have been encountered. Note: In the first version of 1410 ForTran (on processor tapes through version 8) it was possible to equivalence floating point and fixed point arrays (provided the word sizes were the same) thereby saving storage. This is not permitted now. In 1401 ForTran such equivalences are valid.

A General Transformation Program for Multiple Regression Data*

In preparing data for regression analysis, economists often require that the data be transformed in some fashion; ratios, logarithms, lags, etc. are often useful. The transformed data is then used as input for the regression program either instead of, or in addition to, the original data.

In working with Frank de Leeuw on an econometric model of the financial sector of the economy, it was evident that we could benefit greatly from a computer program capable of making a large variety of such transformations; work on the desk calculator occupied a great deal of time and the slowness of this method discouraged much off-the-cuff experimentation. I therefore wrote a general transformation program in ForTran for the IBM 1401. The program requires 16,000 digits of core memory, and, although this prohibits its use at most of the Banks, our experience may be halpful to others contemplating such a program for their machine.

We were working with time series data that was punched into cards in the format required by the Board's multiple regression program. The program is written to read these cards, perform the specified transformations, and punch the results and the original data into a new set of cards in the same format. For each time period the program can read in and punch out up to 60 variables. Control cards are used to indicate the transformations to be performed and the desired position of the transformed variables in the output cards. The transformed variables can be substituted for original variables, or can be punched in addition to the original data.

*Prepared by Ann Walka, Statistical Assistant, Business Conditions Section, Board of Governors.

The program has 10 basic operations. It can add, subtract, multiply, or divide any two variables in the same observation period, take first differences, lag data one period, calculate the ratio X_t/X_{t-1}, find the logarithm, multiply a variable by a constant, and add a constant to any variable.

Because transformed variables may themselves be subsequently transformed, some fairly complicated functions can be obtained. An inconvenience in doing this with the present program is that intermediate results must be included in output. For example, to obtain $3(X^2-Y)^2$ it is necessary to have in the punchout X^2, (X^2-Y), $(X^2-Y)^2$, and finally the desired $3(X^2-Y)^2$. A transformation program which suppresses immediate results might be useful if complicated transformations are to be made frequently.

From our experience with the program, we find that the operation that has been seldom, if ever, used is the ratio X_t/X_{t-1}. Operations that we find could be profitably added to the program are a general lag routine where the number of periods would be specified, a percentage change routine, and an antilog routine.

Because the transformation program is written in ForTran and the output was desired in punched card form for the regression program, it has benefitted greatly from a Fortran function written by Herb Schwartz which causes signs to be punched over the low order digit of the output rather than to the left of the high order digit as is normal with Fortran. This function would be valuable to anyone else writing a similar program in Fortran.

PROGRAMMING THE 1401 DIVIDE FEATURE*

When the beginning programmer is told that his organization has the Multiply-Divide feature on its IBM 1401, he is inclined to think that this means it is going to be easy to multiply and divide. He is shocked from this false sense of security when he is first exposed to the rules for programming a division with the use of the feature. After he is on the job, he is likely to continue to be annoyed by the complexity of programming a division and frustrated by the common occurence of programming errors in one part or another of his division routines.

It is something of a paradox that programming a division may actually be easier at an institution that does not have the Multiply-Divide feature. At that institution, the programmers first write themselves a division subroutine that is then available for use in all subsequent programs. With the typical subroutine of this sort, one moves the dividend and divisor to a particular place, branches to the subroutine, and gets the quotient. With such a subroutine, division becomes, in the words of a programmer at the Fed in Philadelphia, a snap.

For some unknown reason, on the other hand, programmers for machines with the special feature have generally been left to their own resources. This note is intended to remedy this injustice. It discusses the computer instructions that must be given to perform a division correctly, and at the same time presents a form (division worksheet) that can be used to program a 100 per cent accurate division the first time every time.

To write a division operation, a programmer always starts with at least seven items of information. He must know the name or core location of the dividend and divisor; the length of the dividend and divisor fields; the number of decimal places, if any, in the dividend and divisor; and the number of decimal places, if any, desired in the quotient. Suppose we take a typical problem as an example. We want to divide a 5-digit number with 2 decimal places by a 10-digit number with no decimal places, and get the answer rounded to 4 decimal places. The dividend is the change in deposits at banks and so can be either positive or negative, which means that the quotient can also be either positive or negative.

The items of information that the programmer has in this case can be listed as follows:

	Name of Field	Length of Field	Decimal Places
Dividend	DVD	5	2
Divisor	DIV	10	0
Quotient			4

*Prepared by Emanuel Melichar, Economist, Division of Research and Statistics, Board of Governors. With appropriate modifications in mnemonic operation codes, this discussion is also applicable to the IBM 1410.

The first step in using the division worksheet is to write these items into the places provided at the top of the page, as has been done in the excerpt from the worksheet that is shown below. Note that the five items of numerical information have each been assigned a letter of the alphabet by which they may be more easily referred to later in the worksheet as well as in this discussion. Thus, for instance, the length of the dividend field will be referred to as A, the length of the divisor field as B, the number of decimal places in the dividend as C, and so forth.

	Name of Field	Length of Field	Decimal Places
Dividend	DVD	A = 5	C = 2
Divisor	DIV	B = 10	D = 0
Quotient			E = 4

With this information recorded on the worksheet, the sheet can become a useful part of the documentation of the program for which the division routine is being written. Note that space is also provided for filling in the name and length of the field into which the programmer will move the quotient. Often, however, such information is not pertinent because the quotient is moved directly into an output area.

The next part of the division worksheet, labeled "Computation of Character Adjustments," consists of a set of simple calculations which yields answers to such questions as how big an area is needed for the field in which the division will take place, where the dividend should be placed in this field, which digit of this field should be addressed by the B-operand of the divide instruction, and so forth. All of these calculations, however, have been reduced to simple and foolproof formulas.

The formulas as they appear on the worksheet are shown below. The values to be used in the calculations, and also the answers, are referred to by letters of the alphabet.

Computation of Character Adjustments

G = D + E - C = _____

H = _____ (If G is greater than 0, enter that value for H. If not, enter 0.)

F = A + B = _____

J = F + H + 2 = _____

K = B + 1 = _____

L = A + G = _____

M = A + H + 1 = _____

N = L - 1 = _____

The first two formulas call for computing the values of G and H by following the directions. To get the value for G, one must add the values of D and E and subtract the value of C. In our example, D is zero, E is 4, and C is 2. Adding D to E and subtracting C therefore yields 2, which is the answer for G that is filled into the blank space provided on the worksheet.

The directions for filling in the value of H state that H is to be the same as G if G is greater than zero. This is the case in our example, so we fill in the value of 2 in the space provided. If G had been zero or negative, we would have filled in a zero for H, according to the directions on the worksheet.

The values of the other items are similarly calculated according to the formulas given. After this is done, the results for our example appear as follows:

Computation of Character Adjustments

G = D + E - C = 2
H = 2 (If G is greater than 0, enter that value for H. If not, enter 0.)
F = A + B = 15
J = F + H + 2 = 19
K = B + 1 = 11
L = A + G = 7
M = A + H + 1 = 8
N = L - 1 = 6

The next part of the worksheet contains instructions for setting up the division area and the constants that are needed. The field in which the division will take place will be referred to by the label BF given to its high order position, and the length of this field must be at least J+1, which is 20 in our example. The worksheet shows one way in which such a field can be set up in either SPS or Autocoder, namely, by establishing a one-digit DCW labeled BF, followed by a DC with a count or length at least as great as J, which is 19 in our example. This part of the worksheet is shown below with the count for the DC filled in as 19:

Required Constants

Count	Label	OP Code	Contents	Comments
1	BF	DCW		
19		DC		Count must be at least as great as J.
1	ZERO	DCW	0	
1	FIVE	DCW	5	

The length of the field can be greater if this is convenient, but it must be at least as great as the value calculated for J+1 for any division that is to be performed in that field. The only other requirements are that the field have a word-mark in the high order position and that this position be labeled BF. These requirements are automatically met by use of the DCW command as indicated above.

Two constants are also required, a zero and a five. Chances are that these constants have already been set up for use elsewhere in the program. If not, they must be established for use in the division routine. Programmers

writing in SPS can follow the example in the worksheet, while those writing in Autocoder will undoubtedly prefer to use numeric literals rather than the DCW method to get the constants.

The final step is writing the seven instructions required to perform the division. This is done by filling in the blanks in the instructions printed on the worksheet and then copying these instructions onto the coding sheet for the program. This part of the worksheet appears as follows:

Instructions Number	OP Code	A-Address	B-Address
1	ZA	ZERO	BF + _____ (J)
2	ZA	_____ (Dividend)	BF + _____ (F)
3	MZ	BF + _____ (F)	BF + _____ (J)
4	MZ	ZERO	BF + _____ (F)
5	D	_____ (Divisor)	BF + _____ (K)
6	A	FIVE	BF + _____ (L)
7	MZ	BF + _____ (M)	BF + _____ (N)

The Quotient is located at BF +N, is signed, and in L digits in length.

Note that worksheet makes the writing of the instructions utterly simple because all that the programmer now has to do is to fill in the blanks with the appropriate numerical values previously calculated. These values become character adjustments in the instructions. For instance, in our example the B-operand of the first instruction becomes BF with a character adjustment of +019 if written in SPS. In Autocoder it is BF+19. In either case, the writing of the instruction has been "automated" and the chance of making an error has been drastically reduced. The same procedure is followed in writing the remaining instructions, with corresponding benefits. The complete set of instructions in our example, ready for copying to a coding sheet, is as follows:

Instructions Number	OP Code	A-Address	B-Address
1	ZA	ZERO	BF + 19 (J)
2	ZA	DVD (Dividend)	BF + 15 (F)
3	MZ	BF + 15 (F)	BF + 19 (J)
4	MZ	ZERO	BF + 15 (F)
5	D	DIV (Divisor)	BF + 11 (K)
6	A	FIVE	BF + 7 (L)
7	MZ	BF + 8 (M)	BF + 6 (N)

The Quotient is located at BF+N, is signed, and in L digits in length.

If one stops to consider the procedures by which this set of instructions have been obtained, one realizes that it has been completely unnecessary for the programmer to know anything about the complicated rules for doing a division. This is as it should be, because then temporarily forgetting one of the rules does not result in an error in the program.

A brief discussion of the function of each of the instructions, however, may be informative to neophyte programmers. Instruction (1) puts zeros into the field in which the division will take place and places AB bits in the low order position of this field. It thus prepares for the division operation.

Instruction (2) moves the dividend into the correct location in the division area. After it is moved in, the number of spaces remaining at the high order end of the division area will always be equal to the length of the divisor field plus 1, as is required. At the low order end of the division area, spaces are automatically allowed for the development of extra digits in the quotient, if any are required (two were needed in our example), and two additional spaces have also been allowed. One of these will be used in rounding the quotient and the other is used to retain the sign of the quotient during the rounding operation. In addition, the zero and add op code of instruction (2) has insured that the low order position of the dividend in the division area is now signed with either a B-bit or AB-bits.

Instruction (3) now moves these zone bits to the low order position of the division area. It is necessary that there be either a B-bit (if the dividend is negative) or AB-bits (if the dividend is positive) in the low order position of the division area because the computer looks for these bits during the execution of the divide instruction in order to discover when the division is finished. It does <u>not</u> look for a word-mark to find out this fact. This has two implications. First, as previously noted, the length of the area used for division can be longer than J+1 as long as there is a word-mark in the high order digit and the high order end of the area is used for the division. Thus the same area can be used for several division routines involving dividends and divisors with different lengths. Second, it is important to ensure that there will be zone bits in the low order position of the division area and nowhere else in the area, which is invariably accomplished by instructions (1) through (4). The lack of zone bits to stop the division operation correctly can cause a very puzzling kind of error stop.

Instruction (4) removes the zone bits from the low order position of the dividend in the division area. Failure to remove these bits would cause the division operation to stop too early.

Instruction (5) is the actual division operation. As required, the B-operand of this instruction addresses the high order position of the dividend in the division area.

Instruction (6) rounds off (half-adjusts) the quotient by adding 5 to the proper location. As noted above, an extra digit has been developed so that the sign of the quotient will not be stored in the location to which the 5 is added, because then negative quotients would be rounded in the wrong direction. This avoids an apparently common error condition that is hard to detect during testing and that has even been observed to be present in supposedly debugged programs.

Instruction (7) moves the zone bits to the low order position of the properly rounded quotient. The quotient will therefore always be signed with either a B-bit or AB-bits.

The complete division worksheet is reproduced on the following page. At the botton of the sheet, there are instructions to cover special cases in which the number of instructions in the division routine can be reduced, including detailed instructions for the common special case in which the dividend field is unsigned (no zone bits) and it is known that the quotient will always be positive.

Additional copies of the division worksheet are available from the author.

401 Subroutine
Melichar

DIVISION WORKSHEET

For 1401 Multiply-Divide Feature

Program_____ Date_____
Routine_____

	Name of Field	Length of Field	Decimal Places
Dividend	_____	A = _____	C = _____
Divisor	_____	B = _____	D = _____
Quotient	_____		E = _____

Computation of Character Adjustments

G = D + E - C = _____
H = _____ (If G is greater than 0, enter that value for H. If not, enter 0.
F = A + B = _____
J = F + H + 2 = _____
K = B + 1 = _____
L = A + G = _____
M = A + H + 1 = _____
N = L - 1 = _____

Required Constants

Count	Label	OP Code	Contents	Comments
1	BF	DCW		
___		DC		Count must be at least as great as J.
1	ZERO	DCW	0	
1	FIVE	DCW	5	

Instructions

Number	OP Code	A-Address	B-Address
1	ZA	ZERO	BF + _____ (J)
2	ZA	_____ (Dividend)	BF + _____ (F)
3	MZ	BF + _____ (F)	BF + _____ (J)
4	MZ	ZERO	BF + _____ (F)
5	D	_____ (Divisor)	BF + _____ (K)
6	A	FIVE	BF + _____ (L)
7	MZ	BF + _____ (M)	BF + _____ (N)

The Quotient is located at BF +N, is signed, and in L digits in length.

Special Cases

1) If the Dividend is known to be unsigned, you may use MCW for OP code of Instruction (2) and omit Instructions (3) and (4).
2) If the Dividend is known to be signed and positive, you may omit Instruction (3)
3) If the Quotient is known to be positive, you may omit Instruction (7). If you do this, the Quotient will always be unsigned.

Instructions for the common combination of special cases (1) and (3):

ZA	ZERO	BF + _____ (J)
MCW	_____ (Dividend)	BF + _____ (F)
D	_____ (Divisor)	BF + _____ (K)
A	FIVE	BF + _____ (L)

The Quotient is located at BF +N, is unsigned, and is L digits in length.

PUNCHED CARD OPERATIONS IN 1401 FORTRAN*

This material, prepared for reference and training purposes, spells out in considerable detail the reading and punching of cards by 1401 ForTran programs. It is assumed that the reader is acquainted with the principles of punched card design and that he has some minimum familiarity with the ForTran language.

Data appears in punched cards in "external notation," and appears in working storage of the computer in "internal notation." Numerical data may be in two basic modes, whole numbers--integers--and numbers with decimal points--floating point numbers. There is a single form of external notation for integers and there is a single form of internal notation for such numbers. There are two forms of external notation for floating point numbers and there is a single form of internal notation for such numbers. This material will describe, in tabular form, the basic read and punch operations and the techniques (a) for converting external to internal notation on input and (b) for converting internal to external notation on output. There will also be discussions of floating point numbers and the rationale for the two forms of external notation for floating point numbers, and there will be a section on end-of-file procedures in connection with card reading and card punching.

---

* Prepared by M. H. Schwartz, Board of Governors.
The presentation will be continued in the next issue of the Newsletter.

READ Statement — READ S, L...

S is the number of a FORMAT statement that describes how the data in consecutive columns of punched cards are to be collected into individual fields and copied into working storage. S must always be expressed as an Integer constant.

L is the list of variables to be copied from cards into working storage.

Examples of READ statements:

```
READ 12, IDENT, KODE, KDATA(1)
READ 37, X, Y, VALI, VAL2
READ 132, IDENT, X, Y, KODE
```

FORMAT Statement — FORMAT ( )

The FORMAT statement does the following: (1) specifies the mode of each variable to be read (fixed point or floating point); (2) specifies the width, i.e., the number of digits in each field and, in the case of floating point numbers, the number of decimal places and the type of conversion required to translate the number into internal notation; (3) specifies columns of the card not relevant to the current operation that are not to be copied into working storage; (4) provides for alphanumeric information to be read from a card for printing or punching of captions; and (5) indicates the number of cards to be encompassed within the READ.

Examples of FORMAT statements:

```
 12 FORMAT (I8, I1, 12X, I6)
 37 FORMAT (F3.1, F5.3, 16X, 2F4.1)
132 FORMAT (I8, F3.1, F5.3, 51X, I1)
```

List

The List consists of names of simple variables and subscripted variables in the sequence in which the variables appear on the card. The subscripts may be expressed as fixed point constants or as fixed point variables whose values have been specified, read in, or computed. Variables to be by-passed are not named in the List; the fact that the columns occupied by unnamed variables are to be by-passed is specified in the FORMAT statement by X notation.

Examples of Lists:

```
IDENT, KODE, KDATA(1)
IDENT, KODE, KDATA(6), KDATA(12)
IDENT, KODE, KDATA(I), KDATA(M)
IDENT, KODE, (KDATA(I), I=1, 12)
```

Examples of Lists, concluded:

(KDATA(I), I=J, K)
KDATA [If KDATA has been earlier specified as an array in a DIMENSION statement, then the full array may be read in if specified in the List by name alone; i.e., it is not necessary to specify subscripts to read a complete array.]
IDENT, KODE, X, Y, VAL1, VAL2

A card reading example:

Contents of card, by columns

1-8, Bank number
9-12, Classificatory codes not required for current purpose
13-20, Total assets
21-28, Total deposits
29-60, Data fields not required for current purpose
61-68, Loans, IPC

READ and FORMAT statements

```
  READ 7, ID, KTA, KTD, IPC
7 FORMAT (I8, 4X, 2I8, 32X, I8)
```

PUNCH Statement — PUNCH S, L...

S is the number of a FORMAT statement that describes how the data from working storage is to be punched into output cards. S must always be expressed as an Integer constant.
L is the list of variables to be punched into cards from working storage.

FORMAT Statement — FORMAT ( )

The FORMAT statement does the following: (1) specifies the mode of each variable to be punched (fixed point or floating point); (2) specifies the width, i.e., the number of columns for each field and, in the case of floating point numbers, the number of decimal places and the type of conversion required to translate the number into external notation; (3) specifies columns of the card that are to be left blank; (4) provides for alphanumeric information to be punched into a card; and (5) specifies the number of cards to be encompassed within the PUNCH.

List

The List consists of names of simple variables and subscripted variables in the sequence in which the variables are to appear on the card. The subscripts may be expressed as fixed point constants or as fixed point variables whose values have been specified, read in, or computed.

Examples of Lists:

```
IDENT, KODE, DKATA(10)
IDENT, KODE, KDATA(6), KDATA(12)
IDENT, KODE, KDATA(I), KDATA(M)
IDENT, KODE, (KDATA(I), I=1, 12)
(KDATA(I), I=J, K)
KDATA  [If KDATA has been earlier specified as an
        array in a DIMENSTON statement, then the
        full array may be punched out if specified
        in the List by name alone, i.e., it is not
        necessary to specify subscripts to punch
        out a complete array.]
IDENT, KODE, X, Y, VAL1, VAL2
```

	EXTERNAL NOTATION	FORMAT FIELD SPECIFICATION	INTERNAL NOTATION
Integer values, Input cards	Whole numbers are called integers in ForTran. Sometimes they are called fixed point numbers.	Each Integer field to be read is indicated by I-field specification of the type Iw.	
	Integers may appear in input cards in variable field widths, with the number of columns dedicated (assigned) to a field equal to the number of positions taken up by the largest value to be contained in the field.	In the I-type specification for each field, w is the width of the field, i.e., the total number of columns dedicated to the field in the card.	Integers will always be copied into storage as fields equal in width to the single field width specified for integers in the PARAM card for the particular compilation.
	Positive integers may appear without an explicit sign punch, for unsigned integers will always be taken as positive. Negative signs must, and positive signs optionally may, appear either alone in a column to the left of the integer or as a zone over the low order digit of the integer. These variations in sign may be mixed within a card or from card to card.	If the sign is punched in a column at the left, the sign column must be included in the w-count of the field specification.	In storage, the sign will always appear as zoning of the low order digit of each field no matter how the sign is punched. An unsigned number from a card will always be given a positive zone sign in storage.
	Leading digit positions of an integer that is shorter than the field in which it is punched may be filled with leading blanks or leading zeros. If the sign has been punched to the left of an integer that is shorter than its field, the sign may have	Note: Successive identical field specifications need not be repeated. I8, I8, I8 may be written as 3I8. I6, I8, I6, I8 may be written as 2(I6, I8). If, during the input process, the program discovers that the list has more variables than are specified in the FORMAT statement,	The digit position occupied by a leading sign punch and/or by leading blanks or zeros will be filled by leading zeros in storage. If the input field width on the card is less than the width specified in the PARAM card, the field in storage will be filled

EXTERNAL NOTATION	FORMAT FIELD SPECIFICATION	INTERNAL NOTATION
been punched to the far left of the field and be followed by blanks or zeros to the high order significant digit of the integer, or it may have been punched immediately to the left of the high order significant digit and be preceded by blanks or zeros to fill the field.	i.e., "it runs out of Format," formatting will begin again at the beginning of the FORMAT statement. If 10I8 is the entire Format statement, then it may also be written as I8; if 3(I6, I8) is the entire Format statement, it may also be written as I6, I8.	on the left with leading zeros up to the PARAM size. /A word mark will be placed at the digit position at the far left of each field./

	INTERNAL NOTATION	FORMAT FIELD SPECIFICATION	EXTERNAL NOTATION
Integer values, Output cards	All integers in storage will be of the common field width specified by the PARAM card for the particular program.	A type Iw specification for each field to be punched, in which w is the total number of columns dedicated to the field in the card.	Integers may be punched out by the program in variable field widths according to the FORMAT specifications.
	All positive integers in storage will have a positive zone associated with the low order digit.		Positive signs will not be punched into the output card.
	All negative integers in storage will have a minus zone associated with the low order digit.	If an integer to be punched out may be negative, a sign column must be provided in the output field and be included in the w-count of the field specification.	A minus sign will always be punched immediately to the left of the high order significant digit of a negative integer.
			Leading columns in the field of an integer whose significant digits are less in number than the number of columns in the field in which the integer is to be punched will be left blank.
Examples	0000000$1^{+}$	I1	1
	0000000$1^{+}$	I2	b1
	0000000$\bar{1}$	I2	-1
	0000379$5^{+}$	I6	bb3795
	0000379$5^{+}$	I7	bbb3795
	0000379$\bar{5}$	I6	b-3795
	0000379$\bar{5}$	I7	bb-3795

	EXTERNAL NOTATION	FORMAT FIELD SPECIFICATION	INTERNAL NOTATION
Integer values, Input cards, Examples (PARAM Specification = 08)	276	I3	00000276+
	+276	I4	00000276+
	276+	I3	00000276+
	-276	I4	00000276-
	276-	I3	00000276-
	bb3795	I6	00003795+
	+bb3795	I7	00003795+
	bb+3795	I7	00003795+
	bb3795+	I6	00003795+
	-bb3795	I7	00003795-
	bb-3795	I7	00003795-
	bb3795-	I6	00003795-
	003795	I6	00003795+
	+003795	I7	00003795+
	003795+	I6	00003795+
	-003795	I7	00003795-
	003795-	I6	00003795-

Note: b = blank column