ILLIAC II MANUAL
USE OF THE NEW ILLINOIS COMPUTER INSTALLATION

Edited by

C. W. Gear

March 1963

DEPARTMENT OF COMPUTER SCIENCE • UNIVERSITY OF ILLINOIS • URBANA, ILLINOIS
CONTENTS

Chapter

1. INTRODUCTION .......................................................... Prepared by C. W. Gear

2. THE OPERATING SYSTEM ............................................. D. B. Gillies

3. A MACHINE DESCRIPTION AND THE MACHINE LANGUAGE ............. C. W. Gear

4. NICAP, THE ASSEMBLY PROGRAM .................................... C. W. Shepard

5. SYSTEM INPUT-OUTPUT AND AUXILIARY STORAGE .....................

6. AUXILIARY EQUIPMENT ................................................ C. D. Shepard

7. COMPILERS ................................................................

8. THE PROGRAM LIBRARY ................................................

NOTE: These are the plans for the contents of the manual. Not all of this is written at this time. Additions will be distributed as they become available.

Date: 6/3/65
Section: Contents
Page: 1 of 1
Change: 1
ILLIAC II MANUAL
CHAPTER 1. INTRODUCTION

TABLE OF CONTENTS

1.1 Introduction
1.2 Use of This Manual
1.3 ILLIAC II Organization
1.4 Future Changes
1. INTRODUCTION

1.1 Introduction

ILLIAC II, the new University of Illinois computer, was designed and built by the staff of the Digital Computer Laboratory. Preliminary study began in December, 1956, and design in December, 1957. Construction began in 1960. The main processing unit was completed in September, 1962, and it then began functioning with paper tape input/output. The gift from the IBM Corporation of input/output equipment makes it possible to use the power of the very fast arithmetic unit and memory to the full. At the time of its introduction, it brings to the University of Illinois campus a machine whose only competitor in speed of operation is the IBM 7030 computer (STRETCH).

Approximate figures for the various operation times are:

- Multiply 6.6 microseconds
- Floating Point Add 2.5 microseconds minimum, 3.5 microseconds average
- Memory Cycle 1.8 microseconds
- Index Operations 1.0 microseconds

These do not represent the whole story as there is a large amount of overlap among the various tasks. Memory cycles, instruction decoding, indexing, and instruction execution can be occurring simultaneously so that, for example, most indexing operations absorb no effective time at all.

Arithmetic is performed on a 52-bit word which represents a floating-point number with sign, 44 fraction bits and a seven-bit base 4 exponent. This gives an unusually large precision of about 13 decimal digits with a range of about $10^{-38}$ to $10^{+38}$. Sixteen index registers add to the ease and speed of programming. These are stored in a very high-speed memory with a cycle time of about .2 microseconds. This memory can also hold up to eight instructions and four additional words of data which can be used as temporary storage in the execution of a high-speed loop.
Additionally, the machine has the capabilities of performing 32 input/output operations simultaneous with computer operations. The central computer is slowed down only if the memory is busy too often.

Input/output equipment that will be connected to the computer includes 65,536 words of drum backup store, access time 6.8 microseconds (circuitry for this was built at this laboratory); approximately 12 million words of disk file backup store on IBM 1301 disk files; ten IBM 729 Mark VI tape units; an IBM 1401 computer with 600 line-per-minute chain printer, an 800 card-per-minute card reader, and a 250 card-per-minute card punch.
1.2 Use of This Manual

The primary input to this computer is by punched cards, and the purpose of this manual is to describe how these should be prepared. This is not a manual of instruction in programming, but a specification of those features that are available in hardware or software; that is, it is a description of the equipment and programs available. It is assumed that the reader already knows how to program.

The interests of the user should determine which parts of the Manual he reads and in what order he reads them. A person who wishes only to use compilers need read only Chapters 2 and 7. Chapter 2 is concerned with operating procedures and Chapter 7 with compilers. On the other hand, the person writing in assembly language will need to read Chapters 2, 3, 4 and 5 in some detail. Chapter 3 describes exactly what each order does, while Chapter 4 describes the ways in which it can be written and punched on a card. Chapter 5 describes the input/output and auxiliary storage programs that constitute part of the system package. Their use is described in terms of the assembly routine. The compilers will make use of the same input/output routines, but the call sequences will be different and will be described along with the compilers in Chapter 7. Chapter 6 contains descriptions of the miscellaneous auxiliary equipment which is required in any computer installation, such as key punches, reproducers, etc., and Chapter 8 contains the program library. This manual, particularly in Chapters 6, 7, and 8, is not fixed. It will be extended as more programs and equipment are added to the system.

The assembly program, NICAP, available on ILLIAC II is designed to make it possible for the programmer to use the powerful multiple-indexing features of this machine easily. A second purpose is to save the programmer learning all of the details of the complex address constructions that are described in Chapter 3. It is possible to write simple programs for this machine in assembly language without absorbing all of the details of Chapter 3, and it is suggested that the assembly language programmer should not read beyond Section 3.3.8 of Chapter 3, before reading over Chapter 4. Chapter 3 can be used mainly for reference for details of precisely how each order works.
1.3 **ILLIAC II Organization**

The machine can be viewed schematically as containing a floating-point accumulator, a very high-speed memory of eight fast registers, and a main memory of 8,192 words. Index registers are stored in four of the fast registers, packed four per word. Words are 52 bits long; index registers are 13 bits long. A diagram is given in Figure 1. Those registers labelled F0, F1, ..., F7 constitute the very high-speed memory. The boxes labelled input/output represent input/output channels. Each channel may have any number of input/output devices attached to it, although only one device on a channel can be running at one time. All channels, however, can run simultaneously.

The accumulator contains a double-precision number. All adds and subtracts work with this double-precision number to give a double-precision result. Before operations like multiply, the accumulator is rounded to single precision. The details of this are given in Chapter 3. In many cases the programmer will not be concerned with the extra precision available, in which case he need not examine the details of all of the operations. If he always uses the store operation STR (Store Rounded and Normalized) rather than the other store operations, then he can consider that the accumulator is single precision and that all of the operations are single precision, and thus write programs without referring to Chapter 3. He should, however, realize that if he does this he cannot analyze exactly the total rounding error that may be present, but it is true, in general, that an analysis made on this basis will lead to a larger error bound.

A few general rules can be stated that make it fairly easy to program in assembler language. These rules are:

(1) The orders which operate on the floating-point accumulator require an operand. The address in the arithmetic order is usually the location of that operand in the memory or in the fast memory.

(2) This operand is put into the fast register F1 before it is taken into the accumulator. This can be seen from Figure 1.
Figure 1
(3) Exceptions to Rule 1 occur for those operations which use the address directly as data. Examples of this type of instruction are: ADE, Add to Exponent; and LRS, Long Right Shift.

(4) Fast register zero, or F0, receives a number that is to be stored from the accumulator. This also can be seen in Figure 1.

(5) The modifier (Index) order uses the address as the operand. An exception to this rule is LDM, Load Modifier.

From Figure 1 it can be seen that fast registers 4, 5, 6, and 7 can be used for temporary storage of floating point numbers or to hold modifiers. The programmer should adopt his own conventions about the use of these. Generally, it seems more convenient not to use F4, F5, F6, or F7 for floating-point numbers, but to reserve them entirely for use as modifiers. F2 and F3 can be used as temporary storage for floating-point numbers. These two registers are usually adequate for this purpose.
1.4 Future Changes

At the time of writing, no compilers have been programmed for ILLIAC II. The current proposal is to have a version of Algol available in 1964 and then to turn our energies to other languages. Since no definite steps have been taken in this direction, comment from potential users is very welcome.

A number of items that are not yet available are described in this manual. In particular, there are only 4096 words of core storage, only two tape units on line, no disk files and some features of the assembly and I/O programs will not be working immediately. Suitable notations will be made in the chapters in which they are described.

This manual will evolve as the hardware and programs available increase. For this reason, all pages are dated and only section numbered. As changes are made, the pages will be retyped and distributed. Periodically an up-to-date version of the manual, suitably bound, will be issued. Since the manual and programs will change, the Digital Computer Laboratory welcomes suggestions about their context and form.
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Change</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Introduction</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.2</td>
<td>The Core-Load Principle</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.2.1</td>
<td>System Translation and Relocation</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.2.2</td>
<td>Binary Cards</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.3</td>
<td>Batch Processing</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Messages from the Batch Processing System</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.4</td>
<td>The System Library Tape</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.A.1</td>
<td>APPENDIX 1</td>
<td></td>
<td>7/8/64</td>
</tr>
<tr>
<td>2.A.2</td>
<td>APPENDIX 2</td>
<td></td>
<td>7/8/64</td>
</tr>
</tbody>
</table>
2. THE OPERATING SYSTEM

2.1 Introduction

The purpose of the operating system is to maintain an efficient use of machine time by eliminating stops in and between user programs. The operating system sequences call-outs of translators and object programs automatically as directed by system control cards and provides the necessary diagnostic messages. The simplest operating system is the batch processing system which allows only one user on the active area of the machine at any one time. More complicated systems may time-share certain of the machine's facilities in order to gain efficiency.
2.2 The Core-Load Principle

Only one area of core may be active at any one time, that is, only one area may be currently addressable by the main frame of the computer. This area has a maximum size of 8192 words but occasionally, for short programs, may be reduced in size in order to share this memory among several programs. When the system loader puts a program into memory for execution it divides the active memory into four areas according to the type of use. The first three of these constitute the user area; the last is the monitor area and is currently 512 words long. The user may not in general use the monitor area. He has, however, complete freedom to make use of the other three areas in the user area. These areas are:

(a) The COMMON Area

This is in the lower part of memory, normally starting at location 0 and is used to provide a common area for data links between various subroutines and main programs. Because of the awareness of block structure in input/output and auxiliary storage transfers, the programmer can also make use of the common area in order to allocate blocks of data for back-up storage buffers. The common area may be relocated in some instances but only by a multiple of 256 words.

(b) The Program Area

Each of the programs that the user requires are loaded into this area. The program area starts immediately above the common area. The programs are loaded adjacent to each other in the order in which they are processed by the system, followed by the necessary library programs that are called from the system library tape. Programs are relocated by an even number of words only.

(c) Erasable

The erasable storage area is typically used for highly temporary storage such as in subroutines. Use of this area allows the various subroutines to use the same storage cells for their scratch pads, thus saving memory. The erasable area starts immediately above the program. It also is relocated by an even number of words.
2.2.1 System Translation and Relocation

Each program that is received is translated from the source language into a relocatable binary object program which is held in card images until load time. If desired the set of binary cards can be obtained as explained below. These binary cards may then be loaded in place of the source language program. In order to provide for the relocation, four different types of addresses are recognized. They are: absolute, common relocatable, program relocatable, and erasable relocatable.

In order to link the various main programs and subroutines together transfer vectors are used. Each symbolic name which is called by a program, but not defined by that program, results in the compilation of such a vector which occupies one word at the front of the program. Therefore, programmers who are working with absolute addresses must be aware of this additional relocation to their program. In particular they should be aware of the action of the ORG-pseudo operation in assembly language which specifies an address relative to the start of the program including its transfer vectors. Such a program, however, is still subject to the relocation described above.
2.2.2 Binary Cards

A program consists of header cards, program cards and a trailer card (which can also contain program). Columns 11-80 of all cards are in the following standard format:

Column 1  7-9 punch for binary

Columns 10+N, 21+4N, 22+4N, 23+4N and 24+4N for N = 1, 2, ..., 14 represent 14 52-bit words \(W_1, W_2, \ldots, W_{14}\) in the following fashion: Columns 21+4N to 24+4N are the 12 least significant bits of quarter words 0 to 3 respectively of word \(W_N\).

\[
\begin{array}{cccc}
1 & 12 & 1 & 12 & 1 & 12 & 1 \\
\text{Column 21+4N} & \text{Column 22+4N} & \text{Column 23+4N} & \text{Column 24+4N}
\end{array}
\]

Word \(W_N\)

Column 10+N has the responsibility for relocating the quarter words and providing the sign bits of \(W_N\):

\[
\begin{array}{cccccccc}
R_{01} & R_{02} & S_0 & R_{11} & R_{12} & S_1 & R_{21} & R_{22} & S_2 & R_{31} & R_{32} & S_3
\end{array}
\]

Column 10+N (12 bits)

\(S_1\) is the sign bit, \(R_{11}R_{12}\) is the relocation (0 = no relocation, 1 = add \(R_1\) (program area), 2 = add \(R_2\) (common area), 3 = add \(R_3\) (erasable area)) to the ith quarter of word \(W_N\).

For program cards columns 9 and 10 are used for control information.
Column 9

\begin{array}{cccc}
\hline
  & A & B & C & D & E \\
  \hline
\end{array}

4 bits

Control bits = 00

Most significant bit of address of first word

A - E are not used in this case.

Number of words on this card

Column 10 contains the least significant 12 bits of the address of the first word to be loaded from this card.

The last card of the program has a similar format except that the control bits are 01. In this case, bit A, column 8 and bits B and C form a 15-bit address which is the quarter word to which control should be transferred if this is the main program. A main program is the first program if there are no programs with zero entry points; otherwise it is the program with zero entry points. (There should be only one such program.)

Header cards with control bits of 10 for the first and 11 for subsequent ones, give information to the loader about the program. This information is:

- Length of program: 13 bits in bit E and column 10
- Length of common storage: 13 bits in bit D and column 8
- Length of erasable storage: 12 bits in column 7
- Number of entry points: 12 bits in column 6
- Number of CALL vectors: 12 bits in column 5

For each entry point indicated, a word is constructed with the BDC name and the 15-bit address corresponding to it packed in the format:

\begin{array}{cccccc}
\hline
  & 1 & 12 & 1 & 12 & 1 & 12 & 13 \\
\hline
\end{array}

13-bit word address

BDC name (36 bits)

2-bit quarter word address

Date: 7/8/64
Section: 2.2.2
Page: 2 of 3
Change:
The N words corresponding to the N entries occupy the first card, words \( W_1, \ldots, W_N \) and as many cards thereafter as necessary if \( N > 14 \). The next M words on the header cards are the M BCD names of the M CALLed subroutines. These have the same format as entry points except that all bits except for the BCD name are 0.

During loading, the entry points do not cause any words to be generated, but the CALLed subroutines each cause a word containing an unconditional transfer to be generated at the front of the program.
2.3 Batch Processing

Batch processing is achieved by stacking together a number of jobs. Each job is separated by an appropriate card. The programs typically contain four different types of cards.

First must come an ID card. This is to inform the machine of the status of the user. The ID card must contain a blank in column 1. It is similar in all respects to the ID cards currently used by the PORTHOS operating system of the University of Illinois 7094. Next will come one or more system control cards. These cards are characterized by containing a $ sign in column 1. Their purpose is to instruct the system on the nature of the program that is to be run. Among these cards will appear one or more source programs. These may be in any of these available languages or in binary. Finally, if the program uses data, the system control card $ DATA will appear and the remainder of the cards are then assumed to be data to the user program.

The system control cards may contain a number of messages. They may appear on separate cards or several may appear on one card separated by a comma. The message must not extend beyond column 64. The messages accepted by the system are:

NICAP

This card indicates that the next source language program to be read will be in the assembly language. Therefore, the first card following this that does not have a $ sign in column 1 and is not a binary card is assumed to be the first card of an assembly program.

PRINT OBJECT

This tells the system that the user desires that the object program of the next translation be printed. This applies to assembly language as well as any algebraic languages.

PUNCH OBJECT

This tells the system to punch a binary object program for the result of the next translation. Note that PRINT OBJECT and PUNCH OBJECT only apply to the next translation and must be repeated for each separate translation.
GO This indicates that after all translations have been completed the program should be loaded and execution should begin provided that there are no fatal errors.

DUMP This tells the system to give a nonzero memory dump if exit from the user program is made via SYSERR. This exit will happen if any of the standard subroutines are used incorrectly. It can also occur if the user terminates with a CALL SYSERR.

BINARY This does not have to precede a binary source deck; however, if it does then there are to be no more programs in other than relocatable binary.

DATA This card causes the system to terminate the translation phase and begin loading. Any cards hereafter are assumed to be data to be user program. If the user has no data at all this card can be omitted.

$ If a system control card has a second $ immediately after the $ is column 1 the card is reproduced on the listing but causes no other effect. Use of this card should be reserved for comments to the operator via an on-line printed facility which currently does not exist.
2.3.1 Messages from the Batch Processing System

Each of the control cards that is read by the system is printed. If inconsistencies are found appropriate messages are printed followed by a row of asterisks. These messages are in general self-explanatory. At the termination of execution via the SYSERR exit a DUMP may be formed. This dump will list each of the fast registers and the accumulator. MO will have been changed to a l in the process. Also, the location at which the CALL SYSERR was executed will be printed.
2.4 The System Library Tape

This tape is on logical unit one and it contains a table of contents followed by the programs. The table lists those programs which are in the monitor areas and those programs which are on the tape approximately in order of frequency of the currents. The monitor programs also have their addresses listed in the table. Other programs appear in relocatable binary form on the tape after the table. The contents of this tape are listed in Appendix 2 of this chapter.
<table>
<thead>
<tr>
<th>Message</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GO</td>
<td>Must appear before any programs.</td>
</tr>
<tr>
<td>DUMP</td>
<td>Causes a memory dump only if SYSERR is called.</td>
</tr>
<tr>
<td>PRINT OBJECT</td>
<td>Should be used with assembly if listing desired.</td>
</tr>
<tr>
<td>PUNCH OBJECT</td>
<td></td>
</tr>
<tr>
<td>NICAP</td>
<td>No NICAP or compiler program may follow this.</td>
</tr>
<tr>
<td>BINARY</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>Last card read by system. Not necessary if no data.</td>
</tr>
</tbody>
</table>
2.A.2 APPENDIX 2

PROGRAMS ON LIBRARY TAPE

<table>
<thead>
<tr>
<th>Major Name</th>
<th>Minor Names</th>
<th>Subroutines Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATAN1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVD1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXP1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GQU1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAG6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LGT1</td>
<td>LGB1, LGE1</td>
<td>SYSIØ, SYSERR</td>
</tr>
<tr>
<td>LGUN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRINT</td>
<td>PUNCH, READ</td>
<td>SYSIØ, SYSERR</td>
</tr>
<tr>
<td>RKG1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIN1</td>
<td>COS1</td>
<td></td>
</tr>
<tr>
<td>SQR1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSAUX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSERR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSIØ</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYSTEM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Change</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1.1</td>
<td>Machine Features</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Additional Equipment</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2</td>
<td>General Mode of Operation</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Delayed Control</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Advanced Control</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Interplay</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Use of Immediate Access Memory</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.5</td>
<td>Use of Fast Memory</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.2.6</td>
<td>Program Interrupt</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3</td>
<td>Order Code for Floating-Point Arithmetic</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.1</td>
<td>The Floating-Point Accumulator</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Zero and Overflow</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.3</td>
<td>Normalization</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.4</td>
<td>Addition and Subtraction</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.5</td>
<td>Multiplication</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.6</td>
<td>Division</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.7</td>
<td>Round-Off</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.8</td>
<td>Correct Overflow and Detect Zero</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.3.9</td>
<td>Floating-Point Orders</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.4</td>
<td>Orders Which Do Not Involve Floating Point</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Interplay Orders</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Block Reservation Orders</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Advanced Control Orders</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Modifier Arithmetic</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.5</td>
<td>Tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.5.1</td>
<td>Table 1. Address Construction</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Table 2. Special Case Information on Instructions</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Table 3. Order Code Index</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.5.4</td>
<td>Table 4. Order Code Listed Numerically</td>
<td></td>
<td>3/5/63</td>
</tr>
<tr>
<td>3.5.5</td>
<td>Table 5. Additional Mnemonics</td>
<td></td>
<td>3/5/63</td>
</tr>
</tbody>
</table>

Date: 7/9/64  
Section: Chapter 3  
Contents  
Page: 1 of 1  
Change: 1
3. A MACHINE DESCRIPTION AND THE MACHINE LANGUAGE

3.1 Introduction

3.1.1 Machine Features

The computer has the following general characteristics:

Word Length 52 bits
Arithmetic Floating-point (multiply time 6.6 microseconds)
Instruction Length 13 or 26 bits
Address Length 13 bits
Index Registers (also called Modifiers) 16, each 13 bits long
Main Memory 8192 (or $2^{13}$) words of core memory
Memory Cycle 1.8 μsec for each memory
Fast Memory 10 words, 0.2 μsec access time
Back-up Memory Two drums--65,536 words total, 6.8 μsec per word
Ten IBM 729 MK VI magnetic tape units
Two IBM 1301 Disk Files--about 12,000,000 words
Input Punched card 800 cards/minute via on-line IBM 1401
Output Line Printer 600 lines/minute via on-line IBM 1401
Punched card 250 cards/minute
Mode of Operation Parallel, highly concurrent
Special Features Interrupt, memory protection, I/O protection

A typewriter will be connected for system comments. It is not directly available to the programmer. Paper tape I/O is also connected to the machine, but it is only used for engineering tests.
3.1.2 Additional Equipment

Possible later additions may include I/O from remote stations, oscilloscope output and data connections to the 7094 computer and the pattern recognition computer being planned in the Digital Computer Laboratory.

All orders are described below for completeness but those designated with an * cannot be used in normal operation; they will cause an interrupt to the system program stored in the high end of memory.
3.2 General Mode of Operation

This and subsequent sections describe in detail the operation and address construction of each order. For most applications it is not necessary to know the details of the address construction since this is handled by the New Illinois Computer Assembly Program (NICAP) described in Chapter 4.

The principal controls and data paths are shown in Figure 1. There are three main control units in this computer called Delayed Control, Advanced Control and Interplay.
Figure 1
Data Paths for ILLIAC II

Date: 3/5/63
Section: 3.2
Page: 2 of 2
Change:
3.2.1. Delayed Control

Floating-point arithmetic is performed in a double-precision accumulator in the arithmetic unit under the control of Delayed Control. IN and OUT are word registers which contain respectively the operand for the next Delayed Control instruction, and the result of the last Delayed Control store order.
3.2.2 **Advanced Control**

Every instruction is obeyed first by Advanced Control. Some instructions, such as those which only change the value of a modifier register (i.e., index register) are obeyed in their entirety by Advanced Control using the 13-bit address arithmetic unit. For orders obeyed by Delayed Control, Advanced Control must form any address required, obtain any operand required and place it in the IN register in advance of the instruction execution by Delayed Control, and store any result from OUT after the instruction has been obeyed by Delayed Control. Advanced Control must also do the following:

(a) Transfer words of instructions from core memories into two registers called F8 and F9 in the fast memory.

(b) Sequence the control counter to define the core address and position inside the word of the present instruction.

(c) Prepare instructions destined for Interplay.

(d) Time-share the core memories with Interplay.

(e) Program interrupt, to be explained later.
3.2.3 Interplay

The basic Interplay operation is a block transfer between the core memory and any one of the input/output devices or back-up memories. This operation requires one interplay channel, which contains counters, word-assembly equipment and provision for accessing core memory and sensing the end of a block transfer. Most devices have their own private Interplay channel. In the case of magnetic tape units, there will be several units associated with a channel, at most one connected to the channel at any one time. Any number of channels may be running simultaneously, and in this case the core memories are time-shared among the various Interplay channels and Advanced Control. For an Interplay order, Advanced Control constructs an address in the address arithmetic unit (AAU) and sends it on to Interplay.

Interplay is responsible for reading and writing blocks of information between core memory and back-up memory of I/O devices, concurrent with arithmetic. For block transfer purposes, the memory may be considered divided into 32 blocks of 256 words each,

<table>
<thead>
<tr>
<th>Block</th>
<th>Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 to 255</td>
</tr>
<tr>
<td>1</td>
<td>256 to 511</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>31</td>
<td>7936 to 8191</td>
</tr>
</tbody>
</table>

Thus a full block begins at some multiple of 256 and ends just before the next multiple of 256. For transfers to or from drum, an entire block must be transferred at one time. For other devices an initial address not necessarily equal to a multiple of 256 may be used, and Interplay decides that the transfer is over when the next multiple of 256 is reached, or a stop indication is received from the device, whichever happens sooner. For example, a stop can be received from a tape unit at the end of a record.
For input (or playback from drum or file) Interplay assembles characters into words, and periodically competes with Advanced Control for the use of memory to store one word. There is also a prior competition between the various Interplay channels active at the time to see which will have the opportunity of competing for Core Memory. The completion of any block transfer causes Interplay to set an indicator which may cause program interrupt (see below).
3.2.4 Use of Immediate Access Memory

Core Memory #0 The memory containing all even-numbered locations:
0, 2, 4, ..., 8190

Core Memory #1 The memory containing all odd-numbered locations:
1, 3, 5, ..., 8191

NOTE: For the period when only one core memory is attached
to the machine, its locations are numbered 0, 1, 2,
..., 4095, and higher addresses refer to locations in
this memory modulo 4096 so address 4096 refers to
location 0, 4097 refers to 1, ..., 8191 refers to 4095.

Memory protection is accomplished by means of the Block Checker,
a device having 32 indicators (called busy block flipflops), one for
each 256-word block of memory. A block may be set busy (indicator on)
by program because of an Interplay transfer in progress, or for any
other reason. Subsequently all addresses going from Advanced Control to
the core memories are checked to be sure that a block being referred to
is not busy. Reference to a locked-out block when not in the interrupt
mode (see below) is a program error and causes: first, read-out even
if write was requested, and second, program interrupt (see section 3.2.6).
3.2.5 Use of Fast Memory

Registers called F0, F1, ..., F9 are specialized in purpose. F8, F9 contain words of instructions currently being obeyed by the computer. F8 holds the contents of some even-numbered location from Core Memory #0, and F9 holds the contents of the next higher numbered memory location: its address is odd so it came from Core Memory #1. Each of these registers is subdivided into four 13-bit fields called control groups. An instruction is made up of one or two control groups, and is classified as short or long respectively. Reading from left (most significant) to right in a word, the control groups are numbers 0, 1, 2, 3. A long instruction occupies any two consecutive control groups in memory, without restriction. Thus word 2000, control groups 0 and 1 could hold a long instruction which would be obeyed from F8,0 and F8,1. Likewise 2000,3 and 2001,0 could hold a long instruction executed from F8,3 and F9,0 and locations 2001,3 and 2002,0 could hold a long instruction executed from F9,3 and (after automatic refill of F8 and F9) from F8,0. With the exception of the instructions CJF, CJS, to be explained later, the programmer cannot refer explicitly to F8 and F9. Their use is automatic.

NOTE: For the period when only one core memory is attached to the machine, instructions are obeyed from F9.

F4, F5, F6, F7 are four registers which may sometimes be considered as full word registers or, more commonly, each is divided into four 13-bit fields called modifiers. These modifiers are numbers M0, M1, ..., M15 and, reading from left to right

F4 comprises M0, M1, M2, M3
F5 comprises M4, M5, M6, M7
F6 comprises M8, M9, M10, M11
F7 comprises M12, M13, M14, M15

So, for example, the instruction CAM 7, 15 (clear add modifier = CAM) would replace the rightmost 13 bits of F5 by the integer 15, and the
instruction STR F6 would normalize, round off and store the contents of
the floating-point accumulator into register F6, thereby overwriting
modifiers M8, M9, MLO, MLI. In the latter case M8 will have most signifi-
cant digit equal to 1 if the accumulator is negative, equal to zero if
the accumulator is positive, and would be composed of all zeros if and
only if the accumulator held zero.

F2, F3 are temporary storage registers used for constants or
intermediate floating-point results.

F1 (also called IN) and F0 (also called OUT) are closely
associated with the floating-point arithmetic unit, in the following way:
Advanced Control pre-processes every instruction obeyed by the machine.
Some, such as CAM above, it completely executes itself, using a 13-bit
Address Arithmetic Unit for any arithmetic required. For instructions
causing I/O actions, it constructs an address, and routes the modified
instruction on to Interplay. For instructions involving the floating-
point arithmetic unit, Advanced Control constructs any address required,
places any operand (obtained from core memory, or fast memory, or from
the address itself) in the register F1, and places the order in a register
called DCR.

Meanwhile, the floating-point arithmetic unit, under Delayed
Control, may be executing a previous instruction. When Delayed Control
is ready to obey this instruction, it copies F1 into an internal register
in the Arithmetic Unit and decodes the order which is held in DCR. Now
F1 still holds the operand read-in so one can say in general that F1
contains the operand used by the last D.C. (Delayed Control) instruction.
Therefore, for example, one could square the contents of memory location
200 with the program.

    CAD    200    (Clear accumulator, add (200))
    MPY    F1     (Multiply by last operand).

Results of Delayed Control store orders are placed in F0 and subsequently
copied by Advanced Control to their correct destinations. If the stated
destination is F0, then no further copying is necessary. Thus F0 contains the last number stored from the Arithmetic Unit. Two further instructions LFR (Load Fast Register from core memory) and SFR (Store Fast Register into core memory) are needed to complete the description of what is legal and what is illegal in the use of F0 and F1. Fast registers may be used for operands as follows:

1. Delayed Control operands may come from any of F0 through F7 or from core memory.

2. Delayed Control results may go to F0, F2, ..., F7 or core memory, but not F1.

3. LFR can load F2, ..., F7 but not F0 or F1.

4. SFR can store F0, F2, ..., F7 but not F1.
3.2.6 Program Interrupt

Under certain conditions, some of which have already been described, it is desirable to break into a program and execute a different program retaining the option to resume the old program from point of exit. The action of leaving the program and retaining such information as is required to resume it later is called program interrupt. Program interruption causes the transfer of control to the system program area of memory where the necessary fix-up is performed. Causes which might justify interrupt include the following:

1. Correctable machine malfunctions, such as the incorrect read-in of a block from a magnetic tape unit. In this case it is very possible that a second reading of the same section of tape can be done error-free, and it is convenient to have the system program handle this correction automatically for the programmer.

2. The completion of a block transfer or tape rewind, etc. In this case the system program may wish to give another block transfer to Interplay if it has been so instructed by the programmer.

3. Illegal order executed by Advanced Control. During program debugging it is desirable to print the location and contents immediately rather than allowing control to proceed, perhaps by obeying data as instructions. In a production run the occurrence of an illegal order means either a machine malfunction or that the program was not properly debugged.

4. An unusual and possibly unwanted arithmetic result, such as floating-point overflow.

5. Periodic real-time signals furnished by a clock. This permits a system program to supervise code checks, and possibly keep a log, etc.
(6) Any I/O order (PID, POD, IBT, ASN, SSN, or SSR). When the system is present, all I/O must be done by system subroutines in order to provide for I/O protection.

After interruption has taken place, the machine operates in a different mode called the interrupt mode, until a particular order is obeyed (JDC with B = 0 to be explained later). In this mode orders referring to Interplay and to the Block Checker are made legal, all references to busy blocks are legalized and no further interruptions may take place. An interrupt program determines the cause of the interruption, takes appropriate steps to remedy the situation, and, if possible, resumes the program with a JDC, B = 0 order which takes it out of the interrupt mode and back to the program.
3.3 Order Code for Floating-Point Arithmetic

A word consisting of instructions is divided into four 13-bit fields called control groups. An instruction consists of one or two control groups and is referred to as short or long respectively. A short instruction has three fields reading from left to right or most significant to least significant.

F Seven bits designating the operation to be performed. These bits may be designated by a three-letter mnemonic such as MPY for multiply, or by three octal digits. For example, MPY is 120 in octal or 1 010 000 in binary.

B Four bits usually designating a modifier register \( M_B \) or a fast register \( F_B \).

C Two bits which usually control address or operand preparation and may indicate whether the instruction is short or long.

A long instruction consists of \( F, B, C \) in one control group, and a second control group, called \( N \) which is usually an address.

Instructions destined for Delayed Control fall into four categories:

- **Full-word Arithmetic** (such as MPY)
- **Full-word Store** (such as STR: normalize, round and store)
- **Exponent Arithmetic and Shifts** (such as ADE: add to exponent)
- **Quarter-word Store** (the orders SLA: store integer part as an address; and SEX: store exponent).

Consider first the interpretation of \( B, C \) and possibly \( N \) for full-word arithmetic:

**If** \( C = 0 \), modifier \( M_B \) contains an address \( (M_B) \) of a core location containing the operand.

**If** \( C = 1 \), again \( (M_B) \) defines the core location containing the operand, but also \( 1 + (M_B) \) is returned to \( M_B \).

**If** \( C = 2 \), a long instruction with core address \( N + (M_B) \) mod 8192.

**If** \( C = 3 \) and \( B < 8 \) the operand is contained in fast register \( F_B \).
If $C = 3$ and $B = 8$ the core address is $N$.

If $C = 3$ and $B = 9$ the integer $N$ converted to floating point is itself the operand. In this case the leftmost digit of $N$ is considered to have negative weight so $-4096 \leq N \leq 4095$.

If $C = 3$ and $B = 10$ the fraction $N$ converted to floating point is the operand, and $-1 \leq \text{operand} \leq 1 - 1/4096$.

If $C = 3$, $11 \leq B \leq 14$. Unassigned. At present has the effect that floating point zero is the operand, but these should not be used in programs because later additions to the computer might require the use of these combinations.

If $C = 3$, $B = 15$, floating-point zero is the operand.

Any order may be changed if it is preceded by an "add to next" type order, such as ATN, SFN, ASN, SSN, or a modifier arithmetic order with $C$ field equal to 1 or 3. In this case the address of the present order, if any, is affected by the preceding order. For floating-point orders only $C = 3$ and $B < 8$ or $B \geq 11$ are unaffected by a preceding "add to next" type order.

In summary, the instruction is short unless $C = 2$, or $C = 3$ and $B$ is one of 8, 9, or 10; it refers to core memory if $C < 3$, or $C = 3$ and $B = 8$; it refers to fast memory if $C = 3$ and $B < 8$; and Advanced Control constructs an operand from the $N$ address if $C = 3$ and $B = 9$ or 10, or supplies the (zero) operand if $C = 3$, $B = 15$. If $C = 1$ counting is performed on the modifier register specified. Since there are 16 active modifier registers and not 15, the case $C = 3$, $B = 8$ is necessary to specify a fixed-memory location. The computer may be expected to run somewhat faster if short orders are used instead of long ones, and if registers in fast memory are used in preference to locations in the core memory.

For full-word store orders the core memory address or fast memory address specifies a destination rather than a source and the cases $C = 3$, $B = 1$ or $B \geq 9$ are illegal. (In the description of the fast memory it was stated that it was illegal to store into Fl, so $C = 3$, $B = 1$ is illegal here. For $C = 3$, $B \geq 9$ an operand destination is meaningless.
For exponent arithmetic, the "core address" defined above is not used to go to core memory, but rather is reduced to eight bits and combined with the exponent. More exactly, a word consisting of four copies of the address is placed in the IN register and Delayed Control combines arithmetically the right-hand eight bits of this word with the exponent. Shift orders are also included in this class; however, only the rightmost seven bits are used to define the number of shifts. The cases C = 3, B ≠ 8 are illegal for exponent arithmetic orders and shift orders.

For the quarter-word store orders, the B digits define the modifier register destination. These orders cannot refer to core memory, and C is irrelevant. The instruction SIA should have B = one of 0, 4, 8, 12 because the integer will appear in the first 13 bits of the OUT register. The instruction SEX should have B = one of 3, 7, 11, 15 because the exponent will appear in the last 13 bits of the OUT register. If other B combinations occur they are not called illegal by the computer and might just be useful. For example, SIA, M1 would cause the 13 bits immediately to the right of the radix point to be stored in modifier #1.
3.3.1  The Floating-Point Accumulator

There are a number of registers in the arithmetic unit whose action is required in the execution of instructions, which need not be described in the order code because results do not end up there. For one order, SRM, we shall have to refer to some of these extra registers, but otherwise the description will center around the basic registers which hold the results of each instruction.

Accordingly, the accumulator consists of two registers, A, E. A holds 89 bits called $a_0, \ldots, a_{88}$ in twos complement notation, with value

$$a = -a_0 + \sum_{i \neq 0} 2^{-i} a_i, \text{ so } -1 \leq a \leq 1 - 2^{-88}.$$  

The first 45 bits of A ($a_0', \ldots, a_{44}'$), with value $-a_0' + \sum_{i=1}^{44} 2^{-i} a_i$ form a fraction called $a_m'$ ("A Most"). A zero followed by the remaining digits of A ($0, a_{45}', \ldots, a_{88}'$) form a fraction which is sometimes assigned the value $\sum_{i=1}^{44} 2^{-i} a_{44+i}'$ and is called $a_l'$ ("A Least"). These definitions will be used in describing some orders. E holds eight bits called $e_7', e_6', \ldots, e_0'$ with integer value

$$e = -128e_7' + \sum_{i \neq 7} 2^i e_i' \text{ so } -128 \leq e \leq 127.$$  

If a calculated $e$ falls outside this range it is held modulo 256.

Shifts are base 4 only (two binary places at a time), and the exponent $e$ signifies a power of 4. The accumulator holds the number $n = a \cdot 4^e$. Note that $a = a_m' + 4^{-22} a_l'$.

A word $W$ in memory (core or fast) consists of a 45-bit fraction $x$ followed by a seven-bit exponent $y$ at the right-hand end of the word. Its value is $w = x \cdot 4^y$, $-64 \leq y \leq 63$, and fields $x$, $y$ are represented in twos complement notation. Note that the range of exponents permitted in the accumulator is about twice that in memory, and the accumulator holds a double-precision number.
3.3.2 *Zero and Overflow*

The representation is memory of a floating-point zero is $0.4^{-64}$, i.e., zero fractional part and the most negative exponent possible, and it is the only floating-point number with this exponent. When $-64$ is detected as the exponent of an operand, some orders such as ADD (see later) are bypassed. In the accumulator, a zero indicator Z is turned on whenever $a = 0$, or when a calculated exponent is less than $-128$. The contents of the floating-point accumulator is not otherwise altered (it is not cleared to a fixed value) so the numerical value of the accumulator contents depends on Z as well as the contents of A and E. Whenever f is changed, Z is cleared. Store orders, logical shift orders and orders which are bypassed do not clear Z. When the operand of certain arithmetic orders have exponent equal to $-64$, the arithmetic is not done and the order is bypassed.

An overflow indicator OV is turned on whenever any result is too large to be correctly represented and remains on until cleared by a special jump-on-overflow order (JDC with B = 10 or 11). If Z is on, the setting of OV is inhibited except for the inverse divide order (VID), in which case the memory operand divided by the zero accumulator contents is judged to be an overflowed number.

In floating-point arithmetic, overflow of the fractional part is corrected by a right shift of A (division by four) and the addition of one to the exponent. For logical shifts: SRS, LRS, BLS the loss of digits at the left end of A is considered normal. Therefore, for non-store orders, OV is set only if e exceeds 127 or if we are asked to divide by zero.

For store orders one may be required to supply a particular representation of the number, and in this case it turns out that either the fraction or the exponent may overflow the more restricted range of numbers permitted in the memory. In this case OV is also set.

Note that Z gives a continuous indication of whether the accumulator now holds zero, whereas OV is a cumulative indicator telling whether any result has exceeded range since OV was last reset.
The floating-point store orders (including STF: store fixed point) conform to the convention on zero numbers in memory, in that if $Z$ is on, or $e \leq -64$, or the 45-bit fraction to be stored consists of all zeros, then the number $0 \cdot 1_{-64}$ (absolute zero) is transferred to memory.

The conditions $Z$ on or $y = -64$ or $x = 0$ affect the following orders:

- **ADD/SUB**
  - $y = -64$
  - bypass the order
- **ADD/SUB**
  - $z$ on and $y \neq -64$
  - obey "clear add"/"clear subtract"
- **MPY, Z on**
  - bypass the order
- **MPY, Z off, $y = -64$**
  - partial normalize (see later), set $Z$ on, then bypass the order
- **DIV, $x = 0$**
  - set OV and bypass the order
- **DIV, Z on, and $x \neq 0$**
  - set remainder = 0 and bypass the order

When OV has been set, the results in the accumulator are judged wrong, and no attempt is made to maintain a consistent representation of wrong numbers. The orders SAM, SAL, SEX are logical in nature. (They allow the programmer to store the digits in the accumulator without having any floating-point conventions imposed on him.) If he later uses such a number as a floating-point operand it may have exponent -64 and non-zero fractional part.
3.3.3 **Normalization**

A number $p \cdot 4^q$ is called normalized if one of

(a) $Z$ on
(b) $p = 0$
(c) $-1 \leq p < -\frac{1}{4}$
(d) $\frac{1}{4} \leq p < 1$

holds. Since $p \cdot 4^q = (4p) \cdot 4^{q-1}$ a small fraction $p$ may be normalized by repeated left shifts provided one subtracts one from the exponent $q$ for every left shift required. Note that the $Z$ indicator can come on during normalization due to exponent underflow. Except for divide, the results of arithmetic operations are not normalized; however, the accumulator may be normalized at the start of multiply, divide, difference absolute value (DAV) and certain of the store orders.
3.3.4 Addition and Subtraction

The sum $x \cdot 4^y$ and $a \cdot 4^e$ is obtained with an error of at most $4^{-44}$ in its fractional part as follows:

(a) If $|e - y| > 44$, the sum is taken to be the number with the larger exponent.

(b) If $|e - y| \leq 44$, the fractional part of the number with small exponent is right-shifted $|e - y|$ base 4 positions and its first 89 bits (including sign digit) are added to the other fraction. The error, if any, is a truncation error to the right of the 89th bit. The larger exponent is assigned to the result.

NOTE: Some cases of floating-point addition can take a large number of steps by the computer, and a correspondingly long time to execute the instruction. Sometimes these long add or subtract orders can be avoided by careful programming. Relative times for addition can be estimated from the number of steps as follows:

<table>
<thead>
<tr>
<th>Case</th>
<th>$e - y$</th>
<th>Operation</th>
<th>Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 1</td>
<td>$e - y \leq 45$</td>
<td>obey Clear Add</td>
<td>six steps</td>
</tr>
<tr>
<td>Case 2</td>
<td>$-44 \leq e - y \leq 0$</td>
<td>about $5 +</td>
<td>e - y</td>
</tr>
<tr>
<td>Case 3</td>
<td>$1 \leq e - y \leq 22$</td>
<td>about $5 + 2</td>
<td>e - y</td>
</tr>
<tr>
<td>Case 4</td>
<td>$23 \leq e - y \leq 44$</td>
<td>about $-11 +</td>
<td>e - y</td>
</tr>
<tr>
<td>Case 5</td>
<td>$45 \leq e - y$</td>
<td>bypass</td>
<td>three steps</td>
</tr>
</tbody>
</table>

Exceptions. Z true always means Case 1, and Z false but $y = -64$ always means Case 5.
3.3.5 Multiplication

The accumulator is normalized, if necessary, and its first 45 bits are rounded to form a fraction \( a_r \). The product \( x \cdot a_r \) is formed in A, and the sum of the two exponents is placed in E.

If \( a_y = 0 \) normalization is not necessary (and is not done), since the product \( (a \cdot x) 4^{2+y} \) is exact. Likewise if \( a_y \) becomes zero after some even number of base 4 shifts, multiplication begins at that point. Partial normalization may be described by these rules:

(1) If \( Z \) is true or \( a_y = 0 \) or \( a \) is normalized we are done. Otherwise go to (2).

(2) If one left shift (base 4) of \( A \) will normalize \( a \), left shift one place and subtract one from the exponent. If this results in an exponent less than -128 set \( Z \).

(3) Otherwise left shift two places and subtract two from the exponent. If this results in an exponent less than -128 set \( Z \). Now return to (1) above.

The same type of partial normalization is done at the beginning of the DAV instruction.

The rules for ordinary normalization follow:

(1) If \( Z \) is true or \( a \) is normalized we are done. Otherwise go to (2).

(2) If one left shift (base 4) of \( A \) will normalize \( a \), left shift one place and subtract one from the exponent. If this results in an exponent less than -128 set \( Z \).

(3) Otherwise left shift two places and subtract two from the exponent. If this results in an exponent less than -128 set \( Z \). Now return to (1).
3.3.6 Division

First, the accumulator is normalized. Then the number from memory is normalized. If the latter has a zero fractional part, OV is set and the order bypassed at this point. Then if Z is true or the difference of exponents (i.e., the exponent to be assigned to the quotient) is less than \(-128\), the remainder is cleared to floating-point zero, Z is set and the order is bypassed. Otherwise the order is obeyed and a quotient is formed which is either normalized or has fractional part \(-1/4\) and is correctly rounded to 45 bits. After divide \(a_m\) is the fractional part of the quotient, \(a_q\) is zero, and \(e\) is the exponent.

If the Delayed Control order immediately following divide is SRM (store remainder), the remainder from division which was held in other registers in the arithmetic unit called R, ES is transferred to memory. The remainder obeys the floating-point zero convention for numbers to be stored.

Note that divide can produce exponent overflow.

We might call an improper division one in which the normalized divisor has a fractional part smaller in magnitude than the fractional part of the original dividend (before normalization). In this case 47 bits would be required to express the fractional part of the remainder. The first 45 of these are retained, and the two others agree with the 88th and 89th bits of the dividend.
3.3.7 Round-Off

The first 46 bits of the fractional part of the normalized infinite length quotient are rounded to 45 bits adding a one to the 45-bit position, letting carries propagate, and then truncating the result after 45 bits. If the resulting fraction is +1 it is replaced by +1/4 and one is added to the calculated exponent. If this addition of one causes the exponent to become equal to +128, then OV is set.

For orders other than divide, a different procedure is used to obtain the rounded value of a, namely \( a_r \), as follows:

\[
\begin{align*}
  a_r &= a_m \quad \text{if } a_\ell < \frac{1}{2} \\
  a_r &= a_m + 4^{-22}a_{44} \quad \text{if } a_\ell = \frac{1}{2} \ast \\
  a_r &= a + 4^{-22} \quad \text{if } a_\ell > \frac{1}{2}
\end{align*}
\]

The values +1 and -1/4 of \( a_r \) can occur even if a has been normalized. In multiply, inverse divide and difference absolute value, these values of \( a_r \) are used in the arithmetic unit without additional normalization, since it has a somewhat wider range of numbers which may be used during the execution of an instruction. In the case of store orders, the accumulator is not changed after round-off, but the rounded result may be renormalized on the way to the FO register.

---

\* This corresponds to the rule in decimal arithmetic that to round off a five choose the nearest even digit, for example, (.325) rounded = .32 whereas (.335) rounded = .34.
3.3.8 Correct Overflow and Detect Zero

As has been described already, the exponent is monitored during the execution of an instruction, and Z or OV is set if the exponent of the accumulator falls outside of the range \(-128 \leq e \leq 127\). At the end of each instruction which affects the contents of the accumulator, the operation "correct overflow and detect zero" is performed, whose rules follow:

(1) If \(a = 0\) set Z. If Z is set disregard (2) and (3).

(2) If \(-1 > a\) or \(a \geq 1\), right shift A by one place and add one to the exponent.

(3) If (2) results in exponent overflow set OV.

These operations are referred to as "the correction sequence."
3.3.9 Floating-Point Orders

Orders are listed as a mnemonic, followed by a binary plus two octal digit representation of the seven-bit order field F. Where B or C field digits affect the type of order (e.g., JDC orders), other mnemonics can be used in NICAP. These are listed at the end of this Chapter in Table 5 and described further in Chapter 4.

CAD (102) Clear Add. Replace \( a_m \), \( a_e \), \( e \) by \( x \), \( 0 \), \( y \). Z is cleared but would be set if \( x = 0 \) after "the correction sequence."

CSB (100) Clear Subtract. Replace \( a_m \), \( a_e \), \( e \) by \(-x \), \( 0 \), \( y \). Z is cleared but would be set if \( x = 0 \). If \( x = -1 \), then "the correction sequence" replaces \( f \), \( y \) by \( 1/4 \), \( y + 1 \). This could not cause OV to be set since \( y + 1 \leq 64 \).

CAT (103) Clear Add Twice. Replace \( a_m \), \( a_e \), \( e \) by \( 2x \), \( 0 \), \( y \). Z is cleared if \(-1/2 > x \) or \( x > 1/2 \) then "the correction sequence" replaces \( a \), \( y \) by \( 0/4z \), \( y + 1 \). If \( x = 0 \), Z is set and OV cannot be set.

CST (101) Clear Subtract Twice. Replace \( a_m \), \( a_e \), \( e \) by \(-2x \), \( 0 \), \( y \). Z is cleared. If \(-1/2 \geq x \) or \( x > 1/2 \) then "the correction sequence" replaces \( a \), \( y \) by \( 0/4z \), \( y + 1 \). If \( x = 0 \), Z is set and OV cannot be set.

AND (105) Digitwise Logical Multiply. Clear Z. Replace the digits of \( a_m \) with digits consisting of the product \( a_i \cdot x_i \) for each \( i \). Do not change \( a_e \) or \( e \). Z may be set if \( a = 0 \) at the end of this instruction. OV cannot be set nor can corrective right shifts be done.

LOR (106) Digitwise Logical OR. Clear Z. Replace the digits of \( a_m \) with digits consisting of ones wherever \( a_i \) and \( x_i \) are not simultaneously zeros. Do not change \( a_e \) or \( e \). Z is set if \( a = 0 \) at the end of this instruction. OV cannot be set nor can corrective right shifts be done.
**NOT (104)** Clear Add Digitwise Complement. Clear Z and $a_i$. Replace $a_i$ by digits $1 - x_i$ in every digital position of $a_m$. Replace $e$ by $y$. Z will be set if $x$ is composed entirely of ones, OV cannot be set nor can corrective right shifts be done.

**BLS (107)** Single Binary Logical Left Shift of A Most. If $C = 3$, $B < 8$ and $B \neq 1$, $a_m$ is not changed. Otherwise $a_m$ is replaced logically by $2a_m \mod 2$. Z is cleared and will be set if $a$ is 0 or -1. F IN is loaded with an unused operand so to shift use $B = 1$, $C = 3$. To load a fast register into F IN use $C = 3$ and $B < 8$, $B \neq 1$. The shift is not an arithmetic order unless the result is in range.

AND, LOR and BLS are logical, and since they reset Z without replacing the entire contents of the accumulators, should not be used in floating-point arithmetic.

**ADD (112)** Add. Form the sum of $x \cdot 4^y + a \cdot 4^e$ as described on page 1 of section 3.3.4. Apart from the cases $Z$ true of $y = -64$, the accumulator will contain the double-precision sum with exponent equal to the larger of the exponents of the two operands, before overflow is corrected. Z may be set or a right shift of one place may occur, adding 1 to the exponent. This cannot cause OV to be set, since the resulting exponent will not exceed +127. Note that no automatic normalization is done during addition, so it can serve as both floating-point addition and fixed-point addition. The decision on whether to normalize is made at the time of a store order, and depends on the type of store order given.

**SUB (110)** Subtract. Form $(-x) \cdot 4^y + a \cdot 4^e$ in a manner precisely analogous to ADD just above.

---

Date: 7/9/64
Section: 3.3.9
Page: 2 of 9
Change: 1
Multiply.

Partially normalize \( a \cdot 4^e \) and call the result \( a \cdot 4^e \). Then, if either \( Z \) is true or \( y = -64 \), set \( Z \) and bypass the order. Otherwise replace \( a \) by \( x \cdot a \) in \( A \) and \( e \) by \( e + y \) in \( E \). If \( e + y < -128 \) set \( Z \), and if \( e + y \geq 128 \) set \( OV \). Then "the correction sequence" will set \( Z \) if \( x \) was zero and will right shift \( A \) one place and 1 to the exponent if, and only if, \( a = -1 \) and \( x = -1 \). In this case \( OV \) would be set only if \( e + y = 127 \) before shifting.

Divide.

Normalize \( a \cdot 4^e \) and call result \( a \cdot 4^e \). Normalize \( x \cdot 4^y \) and call result \( x \cdot 4^y \). If \( x = 0 \), set \( OV \) and bypass the order. If \( x \neq 0 \) and \( Z \) is true, set remainder equal to zero and bypass the order.

Form \( \left( \frac{a}{x} \right) \) rounded or \( \left( \frac{a}{4^x} \right) \) rounded in \( a_m \), set \( a_y = 0 \) and set \( e \) equal to \( e - y \) or \( e - y + 1 \) respectively. The remainder will have an exponent approximately 22 less than \( e \) unless it is precisely zero. \( OV \) or \( Z \) may be set if \( e - y \) or \( e - y - 1 \) go outside the range -128 to +127. "The correction sequence" will have effect only if \( a = +1 \). In this case the fractional part of the quotient is right shifted one place and 1 is added to the exponent. If exponent overflow results, \( OV \) is set.

Negative Divide.

Identical to DIV except that the divisor is \( (-x) \cdot 4^y \).

Inverse Divide.

The accumulator is normalized and rounded, and \( a_x \) and \( x \) are interchanged and \( a_y \) is cleared; \( e \) and \( y \) are also interchanged. If \( Z \) is true \( OV \) is set and the order by-passed at this point. Otherwise \( x \cdot 4^y \) is normalized. If then \( x = 0 \), \( Z \) is set at this point and the order is by-passed. Otherwise division proceeds from this point, forming \( \left( \frac{x}{a_x} \right) \cdot 4^{y-e} \) or \( \left( \frac{x}{4^a_x} \right) \cdot 4^{y-e+1} \) in a manner analogous to that described under DIV above.
LAL (141) Load A Least. Z is cleared and the digits $a_{45}, \ldots, a_{88}$ are set equal to the non-sign digits of $x$, namely $x_1x_2\ldots x_{44}$. Z would be set if $a_m = 0$ and the non-sign digits of $x$ were all zeros, but "the correction sequence" could have no other effect. This is a logical order and should not be used in floating-point programs.

DAV (122) Difference Absolute Value. The accumulator is partially normalized and $a_r \cdot 4^e$ is formed. It is noted whether or not $Z$ is true at this time. Then $Z$ is cleared and $-|x \cdot 4^y|$ is placed in the accumulator as if by a CAD or CSB order. If now $Z$ was true enter "the correction sequence" with action like that in CAD or CSB. Otherwise enter ADD or SUB to form $|a_r \cdot 4^e| - |x \cdot 4^y|$. Action from this point on is identical to the ADD or SUB order.

STR (124) Normalize Round and Store. If $Z$ is on, store $0.4^{-64}$ in FO and subsequently in the memory location specified. Otherwise normalize. If $Z$ is now true, store $0.4^{-64}$. From this point on the accumulator is not changed, but the number stored may be changed. Form $a_r$. This may still be normalized or it may take on the undesired values $+1$, $-1/4$. In the latter two cases change this to $+1/4$, $-1$ and respectively add 1 to the exponent or subtract 1 from the exponent. If now the exponent is $\leq -64$ store floating-point zero. If the exponent is $\geq +64$ set OV. If floating-point zero is not stored, store the number obtained by the above operations. Since only a 7-bit exponent is stored the exponent is stored modulo 128.
XCH (125) **Exchange.** The new value of the memory register is the same as if STR were executed. The new value of the accumulator is the same as if CAD were executed. Note that for C = 3 the case B = 1 is illegal for this order.

STN (127) **Store Negatively.** The accumulator is normalized and -a_r is transferred to a_m and a_e is cleared. Then STR is executed and it works all right even if -a_r = +1. Following STR, in this case "the correction sequence" would right-shift and add 1 to e.

STF (130) **Store Fixed Point Rounded.** A fixed point number is either $0.4^{-64}$ or has exponent zero. If Z is not on, a $\cdot 4^e$ is converted to fixed point by shifting right or left and counting up or down respectively on e until e becomes zero. If overflow in the fractional part occurs, OV is set and the process continues. After the shift the accumulator is unchanged. If it is equal to +1, OV is set. If Z is true of a_r = 0, $0.4^{-64}$ is stored. Otherwise $a_r \cdot 4^0$ is stored with the exception that +1 is stored as -1. The net result, that numbers are stored modulo 2 except for zero, is called fixed-point representation. Z is set if the result is a = 0.

STU (134) **Store Unnormalized but Rounded.** Identical to STR except that there is no preliminary normalization. The use of this order at key places in a program may considerably reduce the number of shifts prior to store orders and prior to succeeding add or subtract orders.
SRM (143) Store Remainder. If this order follows a DIV, NDV, or VID order with no intervening Delayed Control orders, the remainder (unnormalized, unrounded, but correctly represented even if zero) is stored from the R, ES registers. Since R, ES are used by most instructions, the use of this order following a non-divide order might be useful for engineering routines, but a catalog of results expected would be voluminous. SRM following a divide order sets overflow if the remainder has an exponent \( \geq 64 \), which can happen even when the quotient is in range.

STC (126) Store Clear. The accumulator is not cleared. The first part of this instruction coincides with STR. Suppose \( a_r = a_m + 2^{-42} e \) where \( e = 0 \) or 1. Then \( a_r \cdot 4^e = (a_m + 2^{-42} a_e) \cdot 4^e = [a_m + 4^{-22} (a_e - e)] \). \( 4^e = a_r \cdot 4^e + (a_e - e)4^{-22} \). Now STR stores a number numerically equal to \( a_m \cdot 4^e \). STC after doing this, transfers \( a_e - e \) to \( a_m \), clears \( a_e \) to zero and subtracts 22 from \( e \) so the accumulator holds the remainder from the store operation. \( Z \) is set if \( a_e - e = 0 \) or if \( e - 22 < -128 \). This order allows a double-precision representation in memory in which the most significant half is correctly rounded, and, if STU follows STC, the least significant half has an exponent nearly always 22 less than the most significant half. The exceptions to this rule are when the most significant half rounds to +1 or -1/4, or when the least significant half is judged zero.

ASC (116) Add and Store Clear. Identical in effect to ADD followed by STC. Note that C = 3 and B = 1 is illegal.

SSC (114) Subtract and Store Clear. Identical in effect to SUB followed by STC. Note that C = 3 and B = 1 is illegal.
SIF (131) **Store Integer Part as a Floating-Point Number.** If Z is not true, the accumulator is shifted (see STF) until its exponent becomes equal to +22. This means that the radix point lies 44 bits to the right of \( a_0 \); that is, it lies between \( a_m \) and \( a_4 \). If, during this process, overflow of the fractional part occurs, OV is set. Then if \( a = 0 \) or \( Z \) is true, \( 0.4^{+64} \) is sent to memory. Otherwise \( a_m \cdot 4^{22} \) which is the integer part of the number modulo \( 2^{45} \), is sent to memory. Then "the correction sequence" is obeyed, and it might set \( Z \). Note that if \( 2^{45} \) is the correct answer, the accumulator will have \( -2^{45} \), OV will have been set, and "the correction sequence" will not do a corrective right shift.

SAM (135) **Store A Most.** \( a_m \cdot 4^e \) is transferred to memory, regardless of \( Z \). OV is not set, \( a \) is not changed and \( e \) is stored modulo 128. This is not a floating-point order.

SAL (136) **Store A Least.** \( a_0 \cdot 4^e \) is transferred to memory, regardless of \( Z \). OV is not set, \( a \) is not changed and \( e \) is stored modulo 128. This is not a floating-point order.

SEQ (132) **Store Rounded with Exponent Equal.** This order has no operand, but an operand is implied in FL. Normally, the previous order would have been "Load IN" (LIN) to be described later, but this is not necessary. Whatever number is in FL at the time SEQ is obeyed furnishes a 7-bit exponent, and it is required to shift the accumulator in a manner exactly analogous to STF until its exponent becomes equal to this number and then round-off and store with qualifications identical to STF.
For the next group of orders, Advanced Control places the same address into all four quarters of Fl. This address is interpreted modulo 256 for the first four and modulo 128 for SRS, LRS, and is called $y^s$ and $y^s$ respectively. The most significant bit is a two's complement sign bit.

CAE (117) **Clear Add Exponent.** Place $y^*$ in E. No overflow correction is performed.

CSE (115) **Clear Subtract Exponent.** Place $-y^*$ in E. If $y^* = -128$ set OV.

ADE (113) **Add to Exponent.** Place $e + y^*$ in E. Set OV or Z if the range $-128 \leq \text{exponent} \leq 127$ is exceeded, but do not set OV if Z is true.

SBE (111) **Subtract from Exponent.** Place $e - y^*$ in E. Set OV or Z if the range $-128 \leq \text{exponent} \leq 127$ is exceeded, but do not set OV if Z is true.

SRS (147) **Short Logical Right Shift.** If $y^s$ is positive, translate the digits of the A register right $2y^s$ bits without sign digit duplication and throw away those that pass the right hand end of the $a_m$. If $y^s$ is negative, translate the digits of $a_m$ left, throwing away those that pass the left end of the register. Do not set OV. Z may be set. This is a logical order which would not be useful in floating-point programs. NOTE: $2y^s$ bits are $y^s$ base 4 shifts.

LRS (145) **Long Logical Right Shift.** The double length equivalent of SRS above. This is also a base 4 shift.

Of the Delayed Control orders, there remain only two store orders which produce 13-bit results to be transferred to modifier registers in the fast memory.

SEX (137) **Store Exponent.** A word whose first 39 digits agree with the first 39 digits of $a_e$ and whose last 13 digits agree with the 8-bit exponent $e$ extended five digits left by duplication of the sign digit, is placed on FO, then the 1/4 word of FO aligned with
M₄ is copied into M₄. If the modifier register specified is M₃, M₇, M11, or M15, the last $\frac{1}{4}$, i.e., the exponent, is stored in it. This is a logical order and Z is disregarded. The C field of the SEX order has no effect.

**SIA (133) Store Integer Address.** The accumulator is shifted until its exponent is equal to +6 in a manner similar to SIF. If the base 4 exponent is +6 the radix point lies between the 13th and 14th digits of the A register. OV or Z may be set during the shift but if Z was true beforehand, no shift is made. Now if $a_m = 0$ or Z is true, $0.4^{-64}$ is placed in FO. Otherwise $a_m \cdot 4^e$ is placed in FO. If the modifier register specified is one of M0, M4, M8, or M12 the first quarter word, i.e., the integer part of the accumulator, modulo $2^{13}$, is copied into the M₄. Otherwise a different quarter word is copied (see SEX). The C field of the SIA order has no effect.
3.4 Orders Which Do Not Involve Floating Point

The orders described in the last section are obeyed first by Advanced Control, which obtains any needed operand and places it in Fl, then by Delayed Control which performs any necessary floating-point arithmetic, and then, in the case of store orders, by Advanced Control again. SIA and SEX were a special type: always short, B represents the modifier and C is irrelevant. Otherwise address construction was fairly uniform: if C < 3 or C = 3, B = 8 an address is constructed and depending on the order type this either defines a core memory location, or the address is quadruplicated and used. Let us refer to this process as normal address construction. For floating-point orders additional options were provided: C = 3 and B < 8 means fast register \( F_B \) with the proviso that Fl is not a destination, C = 3, B = 1 is illegal for certain orders. Likewise floating-point operands were generated for the cases C = 3, B \( \geq 9 \). These additional options do not apply to the next class of orders: Advanced Control and Interplay orders with normal address construction. If C = 3 and B \( \neq 8 \) these orders are illegal.
3.4.1 Interplay Orders

*PID (023) **Prepare Input Device.** After the address is constructed it is sent to Interplay and is interpreted as a five-bit field at the left-hand end signifying the channel number and an eight-bit field specifying details of a block transfer into the core memory. At the time of this writing, channel 0 has been assigned to the drum and the eight-bit field represents the drum block. No other assignments have been made yet.

*POD (062) **Prepare Output Device.** Similar to PID above except that a transfer from the core memory is intended.

*IBT (022) **Initiate Block Transfer.** The address constructed specifies the core address at which the transfer will start. Depending on the device in question, the transfer will cease when a stop character is reached or when the core address reaches an address 1 less than the next multiple of 256, whichever happens sooner. In the case of the drum, and probably all devices which operate on a fixed 256-word block, only the first five bits of this address matter—the others are replaced by zeros so a drum transfer always begins at a core address which is a multiple of 256 and ends at the address one less than the next multiple of 256.

PID, POD, IBT are the only orders obeyed by Interplay. They may be used only when Interrupt is disabled. When the system program is in the machine, the user must do his input-output via system subroutines described in Chapter 5.
3.4.2 Block Reservation Orders

*BBF (043) **Busy Block Flipflop.** The first five bits of the address constructed define a block in core memory whose indicator is to be set to the "busy" state.

*FBF (042) **Free Block Flipflop.** The first five bits of the address constructed define a block in core memory whose indicator is to be set to the "free" state.
3.4.3 Advanced Control Orders

LIN (060) Load IN Register. Four copies of the constructed address are placed in F1. This order usually precedes SEQ.

JLH (054) Jump to Left-Hand Control Group. A "jump" instruction is a control transfer or branch operation. JLH has the effect that the next order obeyed begins at the first control group of the core word defined by the address. This is an unconditional jump which lacks generality and is mainly useful in returning from a subroutine.

ATN (021) Add to Next Address. Certain orders, of which this is the first example, influence the address construction of order following. The address formed by this instruction is added to the address of the next instruction and then this next instruction is obeyed normally. ATN may be repeated. Example: suppose one wished to copy into the accumulator the number from the memory location defined by the sum of modifiers M2, M5, M7. The program is

\begin{align*}
\text{ATN } 2,0 \\
\text{ATN } 5,0 \\
\text{CAD } 7,0 \text{ or } 3 \text{ short orders.}
\end{align*}

The first instruction adds \((M2)\) to the second. The second would normally add \((M5)\) to the third, but since it has \((M2)\) added to its address it therefore adds \((M2) + (M5)\) to the third instruction. The third instruction would normally have core address \((M7)\) but this is increased by \((M2) + (M5)\) making a total of \((M2) + (M5) + (M7)\) as required.

SFN (041) Subtract from Next Address. The address formed is subtracted from the address of the next instruction. Note that two SFN orders in a row have the effect of adding the first address and subtracting the second.
ORB (061) **Logical OR with B Digits of the Next Instruction.** The rightmost four bits of the address constructed are combined with the four B bits of the instruction following--the next next instruction will have zeros only in those bit positions of the B field where both bits are zero. This is a useful instruction for breaking up words into quarter words, but is not very useful for combining quarter words into words.

This completes the list of instructions for which normal address construction applies. The next group of instructions are always short.

*ASN (032) **Add Special Register to Next Address.** This is a short instruction. Provision is made in the computer for up to 64 13-bit registers called special registers. The address of this instruction and the two subsequent instructions is the number 4B + C (between 0 and 63). The registers are used as I/O channel condition registers and for special I/O (e.g., to paper tape for engineering and to typewriter for system comments). They cannot be used by the programmer unless interrupt is disabled.

*SSN (072) **Subtract Special Register from Next Address.** This is a short instruction. Similar to ASN except that subtraction rather than addition is done.

*SSR (073) **Store in Special Register.** This is a short instruction, and its address is zero unless SSR was preceded by an "add to next" type order. The B, C fields specify which special register the address is to be stored in. If an instruction such as ATN had preceded this instruction, then, in general, something other than zero would be stored in the special register.
CJF (057) Count and Conditional Jump to First. This is a short instruction. One is added to the contents of modifier $M_B$ and the result is returned to $M_B$. If the result is non-zero, jump to the instruction at F8 position C ($C = 0, 1, 2, \text{ or } 3$); otherwise obey the next instruction in sequence. The purpose of this instruction is to be able to obey simple loops of instructions inside F8 and F9 if the loop condition is just a count. In these cases the instruction words are read from memory just once and are held in F8 and F9 for repeated execution. F8 contains the contents of an even-numbered core memory location, and F9 holds the contents of the next higher-numbered location, which is odd. Note that this implies that the normal method of counting is to place the negative of the count in a modifier register and count up to zero. For a long program consisting of long and short instructions intermixed, the programmer would refer to instructions symbolically and would not, in general, know what word and position any instruction occupied. One of the operations which the assembly routine must be able to do is to insert a jump to the left-hand control group of the next even address so these short loops may be correctly positioned. During the period when there is only one core memory CJF will have the effect of conditionally jumping to F9, position C which means jump to position C of the word containing the present instruction.

CJS (055) Count and Conditional Jump to Second. This is a short instruction, whose action is similar to CJF above except the destination is F9, position C. During the period when there is only one core memory the action of CJS is identical to the action of CJF.
The next group of instructions are always long, and \( N \), the second control group, specifies the address.

**CJU (077) Count and Jump if the Result is Unequal to Zero.** Long. Add one to \( (M_B) \) and return the result to modifier \( M_B \). If it is non-zero jump to word \( N \), position \( C \); otherwise obey the next instruction in sequence.

**CJZ (037) Count and Jump if the Result if Zero.** Long. Add one to \( (M_B) \) and return the result to modifier \( M_B \). If it is zero, jump to word \( N \), position \( C \); otherwise obey the next instruction in sequence. This is a very rarely used instruction—CJU would be much more common in programs.

**JPM (074) Jump if Positive Modifier.** Long. If the leftmost of the 13 bits of \( (M_B) \) is a 0, jump to \( N \), position \( C \); otherwise obey the next instruction in sequence. We may regard the integer held in \( M_B \) as either lying in the range \( -4096 \leq (M_B) \leq 4095 \) if the leftmost digit is regarded as having negative weight or lying in the range 0 to 8191 for core addresses, or -8191 to 0 for orders like CJF, CJJS, CJU, CJZ.

**JNM (034) Jump if Negative Modifier.** Long. If the leftmost of the 13 bits of \( (M_B) \) is a 1, jump to word \( N \), position \( C \). Otherwise obey the next instruction in sequence.

**JZM (035) Jump if Zero Modifier.** Long. If all 13 bits of \( (M_B) \) are zeros, jump to word \( N \), position \( C \). Otherwise obey the next instruction in sequence.

**JUM (075) Jump if Modifier is Unequal to Zero.** Long. If the 13 bits of \( (M_B) \) are not identically zero, jump to word \( N \), position \( C \). Otherwise obey the next instruction in sequence.
JSB (076) **Jump to Subroutine.** Long. Let H be the location of the N address of this JSB instruction. Place H + 1 into M_B and jump to word N, position C. Conventionally B = 3 and the subroutine returns control to the left-hand control group of the word following the JSB instruction. Thus entry to a subroutine at location S would be accomplished by JSB 3, 0, S and return from subroutine would be accomplished by JLB 3, 0.

JDC (056) **Jump on One of a Diversity of Conditions.** Long. The B field specifies one of 16 possible conditions to be tested. If the condition is true, the next instruction is obeyed from word N, position C. Otherwise obey the next instruction in sequence. The conditions are:

- B = 0 Unconditional. This also causes the computer to leave the interrupt mode if it happens to be in it.
- B = 1 Unconditional. This does not change the interrupt status.
- B = 2 Accumulator positive or zero (Z on or a ≥ 0).
- B = 3 Accumulator negative and not zero.
- B = 4 Accumulator unequal to zero (Z not on).
- B = 5 Accumulator zero.
- B = 6 Accumulator positive and not zero.
- B = 7 Accumulator zero or negative.
- B = 8 OV on.
- B = 9 OV not on.
- B = 10 OV on \{ and then clear OV if it was on.
- B = 11 OV not on
- B = 14 Digit a_0 = 0 (useful mainly in logical operations.
- B = 15 Digit a_0 = 1 (useful mainly in logical operations.

Note that if B ≠ 0 or 1 this jump is conditional on the arithmetic result after Delayed Control
has finished any instruction in progress and quite possibly another instruction prepared by Advanced Control. Such JDC orders can greatly slow down the machine, and one of the objectives of good programming is to reduce the number of these, at least in critical parts of a program. B = 12 and B = 13 test an engineering switch and therefore should not be used. (Normally 12 would have the same effect as 1, and 13 would have the same effect as 0.) Other mnemonics may be used for these orders to save remembering the meaning of the B field digits. They are listed at the end of this Chapter.

**LDM (071) Load Modifier from Core Memory.** Long. The quarter word aligned with $M_B$ in word N in memory is copied into $M_B$. If $B = 0, 4, 8,$ or 12 this would be the first quarter word; if $B = 1, 5, 8,$ or 13 this would be the second quarter word, etc.

The remaining instructions are long if $C = 2$ or 3, and short if $C = 0$ or 1. The address is 0 if the instruction is short, and N if the instruction is long. If preceded by an "add to next" type order, the 0 or N is appropriately modified.

**LFR (070) Load Fast Register.** Long if $C = 2$ or 3. Copy the word from core location given by the address into $F_B$. If $B = 0$ or 1 or $B \geq 8$ the instruction is illegal.

**SFR (030) Store Fast Register.** Long if $C = 2$ or 3. Copy the word from $F_B$ into the core location. If $B = 1$ or $B \geq 8$ the instruction is illegal.
3.4.4 Modifier Arithmetic

The remaining 12 orders cause the address to be combined with the contents of a modifier. The result is either returned to the modifier or added to the address of the next order according to the following rules:

C = 0 means that the address is zero (short order) and the result is returned to the modifier.

C = 1 means that the address is zero (short order) and the result is added to the address of the next order.

C = 2 means that a second control group provides an address N (long order) and the result is returned to the modifier.

C = 3 means that a second control group provides an address N (long order) and the result is added to the address of the next order.

Note that if one of these orders is preceded by an "add to next" type order, the 0 or N address is appropriately modified. To avoid writing out the C field explicitly when it is 1 or 3, a second set of mnemonics are listed at the end of this chapter. These have the effect of the associated order described below with an odd C field. They are all derived from the following mnemonics by changing the final M to an N (for "add to next").

CAM (027) Clear Add Modifier. Long if C = 2 or 3. The result equals the address.

CSM (025) Clear Subtract Modifier. Long if C = 2 or 3. The result equals the negative of the address.

ADM (067) Add to Modifier. Long if C = 2 or 3. \((M_B)\) plus the address is the result.

SBM (065) Subtract from Modifier. Long if C = 2 or 3. \((M_B)\) minus the address is the result.
**CNM (024) Clear Negate Modifier.** Long if C = 2 or 3. The digitwise complement of the address is the result. The digitwise complement of a binary number is the number consisting of zeros where the original number had ones, and vice versa. In this case, numerically, the digitwise complement is 8191 minus the address.

**CRM (026) Circular Right Shift Modifier.** Long if C = 2 or 3. The four rightmost bits of the address define a number of shifts p where 0 ≤ p ≤ 15. The result is the modifier contents \(M_B\) rotated right circularly p places. Note that a shift of 13 places brings it back to where it started from so p = 13 has the same effect as p = 0, p = 14 has the same effect as p = 1, p = 15 has the same effect as p = 2.

**ANM (047) AND with Modifier.** Long if C = 2 or 3. The 13 bits of the address are ANDed with the corresponding bits of \(M_B\) to form the result. A bit position of the result has one if and only if both operands had ones in that digital position.

**ORM (046) OR with Modifier.** Long if C = 2 or 3. The 13 bits of the address or ORed with the corresponding bits of \(M_B\) to form the result. A bit position of the result has a 0 if and only if both operands had zeros in that digital position.

**EOM (066) Exclusive OR with Modifier.** Long if C = 2 or 3. The exclusive OR (or addition without carries) of the 13 address bits and the 13 \(M_B\) bits is the result. The result has ones in those bit positions in which the two operands differed.
EQA (064) Equivalent with Modifier. Long if $C = 2$ or $3$. The equivalence function of the address and $(M_B)$ is formed in every digital position of the result. The result has ones in those bit positions in which the two operands agreed.

NAM (045) Negate, then AND with Modifier. Long if $C = 2$ or $3$. The digit-wise complement of the address if formed, and ANDed digit by digit with $(M_B)$ to form the result. The result has ones only in those bit positions where the address had zeros and the modifier had ones.

NOM (044) Negate, then OR with Modifier. Long if $C = 2$ or $3$. The digit-wise complement of the address is formed and ORed digit by digit with $(M_B)$ to form the result. The result has zeros in those bit positions where the address had ones and the modifier had zeros.
3.5 Tables

3.5.1 Table 1. Address Construction

Normal: LAL, CAD, CSE, CAT, CST, NOT, AND, LOR, BLS, ADD, SUB, MPY, DIV, NDV, VID, DAV

Normal, C = 3, B = 1 or B > 9 illegal: STR, STU, STN, STC, STF, SIF, SEQ, SAM, SAL, SRM, ASC, SSC, XCH


Short, B means \( M_B \): SIA, SEX, CJF, CJS  For SIA, SEX C has no effect.

Short, 4B + C is name of special register: *ASN, *SSN, *SSR

Long, C represents 1/4 W except for LDM: CJU, CJZ, JPM, JNM, JZM, JUM, JSE, LDM, JDC

C = 2 or 3 Means Long: LFR, SFR

C = 2 or 3 Means Long, C odd Means Add to Next: CAM, CSM, ADM, SBM, CNM, CRM, ANM, ORM, EOM, NAM, NOM, EQM

* Order is interrupted unless interrupt is disabled.
### Table 2. Special Case Information on Instructions

<table>
<thead>
<tr>
<th>Instruction(s)</th>
<th>Special Case Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAL, CAD, CSB, CAT, CST, NOT, AND, LOR, BLS</td>
<td>Clear Z first</td>
</tr>
<tr>
<td>ADD, SUB, MPY, ASC, SSC</td>
<td>Special cases if Z is true or if ( y = -64 )</td>
</tr>
<tr>
<td>DIV, NDB, VID</td>
<td>Special cases if Z is true or if ( x = 0 )</td>
</tr>
<tr>
<td>STR, STU, STN, STC, STF, SIF, SEQ, SIA</td>
<td>Special cases if Z is true; the operand used is ( 0 \cdot 4^{-64} )</td>
</tr>
<tr>
<td>SAM, SAL, SEX, SRM, SRS, LRS, LIN</td>
<td>Disregard Z</td>
</tr>
<tr>
<td>CAE, CSE, ADE, SBE</td>
<td>Z or OV may be set. If Z is true OV is not set</td>
</tr>
<tr>
<td>XCH</td>
<td>Special case if Z is true ( (0 \cdot 4^{-64}) ). Then clear Z</td>
</tr>
<tr>
<td>PID, POD, IBT, BEF, FBF, ASN, SSN, SSR</td>
<td>Cause Interrupt if in interrupt enabled mode</td>
</tr>
</tbody>
</table>

---

Date: 3/5/63  
Section: 3.5.2  
Page: 1 of 1  
Change:
### Table 3. Order Code Index

<table>
<thead>
<tr>
<th>Order Codes</th>
<th>Section No.</th>
<th>Page No.</th>
<th>Order Codes</th>
<th>Section No.</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD 112</td>
<td>3.3.9</td>
<td>2</td>
<td>LAL 141</td>
<td>3.3.9</td>
<td>4</td>
</tr>
<tr>
<td>ADE 113</td>
<td>3.3.9</td>
<td>8</td>
<td>LDM 071</td>
<td>3.4.3</td>
<td>6</td>
</tr>
<tr>
<td>ADM 067</td>
<td>3.4.4</td>
<td>1</td>
<td>LFR 070</td>
<td>3.4.3</td>
<td>6</td>
</tr>
<tr>
<td>AND 105</td>
<td>3.3.9</td>
<td>1</td>
<td>LIN 060</td>
<td>3.4.3</td>
<td>1</td>
</tr>
<tr>
<td>ANM 047</td>
<td>3.4.4</td>
<td>2</td>
<td>LOR 106</td>
<td>3.3.9</td>
<td>1</td>
</tr>
<tr>
<td>ASC 116</td>
<td>3.3.9</td>
<td>6</td>
<td>LRS 145</td>
<td>3.3.9</td>
<td>8</td>
</tr>
<tr>
<td>*ASN 032</td>
<td>3.4.3</td>
<td>2</td>
<td>MPY 120</td>
<td>3.3.9</td>
<td>3</td>
</tr>
<tr>
<td>ATN 021</td>
<td>3.4.3</td>
<td>1</td>
<td>NAM 045</td>
<td>3.4.4</td>
<td>3</td>
</tr>
<tr>
<td>*BBF 043</td>
<td>3.4.2</td>
<td>1</td>
<td>NDV 122</td>
<td>3.3.9</td>
<td>3</td>
</tr>
<tr>
<td>ELS 107</td>
<td>3.3.9</td>
<td>2</td>
<td>NOM 044</td>
<td>3.4.4</td>
<td>3</td>
</tr>
<tr>
<td>CAD 102</td>
<td>3.3.9</td>
<td>1</td>
<td>NOT 104</td>
<td>3.3.9</td>
<td>2</td>
</tr>
<tr>
<td>CAE 117</td>
<td>3.3.9</td>
<td>8</td>
<td>ORB 061</td>
<td>3.4.3</td>
<td>2</td>
</tr>
<tr>
<td>CAM 027</td>
<td>3.4.4</td>
<td>1</td>
<td>ORM 046</td>
<td>3.4.4</td>
<td>2</td>
</tr>
<tr>
<td>CAT 103</td>
<td>3.3.9</td>
<td>1</td>
<td>*PID 023</td>
<td>3.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CJF 057</td>
<td>3.4.3</td>
<td>3</td>
<td>*POD 062</td>
<td>3.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CJS 055</td>
<td>3.4.3</td>
<td>3</td>
<td>SAL 136</td>
<td>3.3.9</td>
<td>7</td>
</tr>
<tr>
<td>CJU 077</td>
<td>3.4.3</td>
<td>4</td>
<td>SAM 135</td>
<td>3.3.9</td>
<td>7</td>
</tr>
<tr>
<td>CJZ 037</td>
<td>3.4.3</td>
<td>4</td>
<td>SBE 111</td>
<td>3.3.9</td>
<td>8</td>
</tr>
<tr>
<td>CNM 024</td>
<td>3.4.4</td>
<td>2</td>
<td>SBR 065</td>
<td>3.4.4</td>
<td>1</td>
</tr>
<tr>
<td>CRM 026</td>
<td>3.4.4</td>
<td>2</td>
<td>SEQ 132</td>
<td>3.3.9</td>
<td>7</td>
</tr>
<tr>
<td>CSB 100</td>
<td>3.3.9</td>
<td>1</td>
<td>SEX 137</td>
<td>3.3.9</td>
<td>8</td>
</tr>
<tr>
<td>CSE 115</td>
<td>3.3.9</td>
<td>8</td>
<td>SFN 041</td>
<td>3.4.3</td>
<td>1</td>
</tr>
<tr>
<td>CSM 025</td>
<td>3.4.4</td>
<td>1</td>
<td>SFR 030</td>
<td>3.4.3</td>
<td>6</td>
</tr>
<tr>
<td>CST 101</td>
<td>3.3.9</td>
<td>1</td>
<td>SIA 133</td>
<td>3.3.9</td>
<td>9</td>
</tr>
<tr>
<td>DAV 142</td>
<td>3.3.9</td>
<td>4</td>
<td>STF 131</td>
<td>3.3.9</td>
<td>7</td>
</tr>
<tr>
<td>DIV 121</td>
<td>3.3.9</td>
<td>3</td>
<td>SRM 143</td>
<td>3.3.9</td>
<td>6</td>
</tr>
<tr>
<td>EOM 066</td>
<td>3.4.4</td>
<td>3</td>
<td>SRS 147</td>
<td>3.3.9</td>
<td>8</td>
</tr>
<tr>
<td>EQM 064</td>
<td>3.4.4</td>
<td>3</td>
<td>SSC 114</td>
<td>3.3.9</td>
<td>6</td>
</tr>
<tr>
<td>*FBB 042</td>
<td>3.4.2</td>
<td>1</td>
<td>*SSN 072</td>
<td>3.4.3</td>
<td>2</td>
</tr>
<tr>
<td>*IBT 022</td>
<td>3.4.1</td>
<td>1</td>
<td>*SSR 073</td>
<td>3.4.3</td>
<td>2</td>
</tr>
<tr>
<td>JDC 056</td>
<td>3.4.3</td>
<td>5</td>
<td>STC 126</td>
<td>3.3.9</td>
<td>6</td>
</tr>
<tr>
<td>JDB 054</td>
<td>3.4.3</td>
<td>1</td>
<td>STF 130</td>
<td>3.3.9</td>
<td>5</td>
</tr>
<tr>
<td>JNM 034</td>
<td>3.4.3</td>
<td>4</td>
<td>STN 127</td>
<td>3.3.9</td>
<td>5</td>
</tr>
<tr>
<td>JPM 074</td>
<td>3.4.3</td>
<td>4</td>
<td>STR 124</td>
<td>3.3.9</td>
<td>4</td>
</tr>
<tr>
<td>JSB 076</td>
<td>3.4.3</td>
<td>5</td>
<td>STU 134</td>
<td>3.3.9</td>
<td>5</td>
</tr>
<tr>
<td>JMB 075</td>
<td>3.4.3</td>
<td>4</td>
<td>SUB 110</td>
<td>3.3.9</td>
<td>2</td>
</tr>
<tr>
<td>JZM 035</td>
<td>3.4.3</td>
<td>4</td>
<td>VID 123</td>
<td>3.3.9</td>
<td>3</td>
</tr>
</tbody>
</table>

**NOTE:** The page number indicates where the order is defined in the text.

* Order is interrupted unless interrupt is disabled.

Date: 3/5/63  
Section: 3.5.3  
Page: 1 of 1  
Change:
### Table 4. Order Code Listed Numerically

<table>
<thead>
<tr>
<th>Second Octal Digit</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Followed by First Octal Digit</td>
<td>ATN</td>
<td>*IBT</td>
<td>*PID</td>
<td>CNM</td>
<td>CSM</td>
<td>CRM</td>
<td>CAM</td>
<td>A.C. Orders</td>
</tr>
<tr>
<td>02</td>
<td>SFR</td>
<td>*ASN</td>
<td>JNM</td>
<td>JZM</td>
<td>CJZ</td>
<td>JDC</td>
<td>CJF</td>
<td></td>
</tr>
<tr>
<td>03</td>
<td>SFN</td>
<td>*FBF</td>
<td>*BBF</td>
<td>NOM</td>
<td>NAM</td>
<td>ORM</td>
<td>ANM</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>LFR</td>
<td>LDM</td>
<td>*SSN</td>
<td>*SSR</td>
<td>JPM</td>
<td>JUM</td>
<td>JSB</td>
<td>CJU</td>
</tr>
<tr>
<td>05</td>
<td>LIN</td>
<td>ORB</td>
<td>*POD</td>
<td>EQM</td>
<td>SBM</td>
<td>EOM</td>
<td>ADM</td>
<td></td>
</tr>
<tr>
<td>06</td>
<td>SUB</td>
<td>SBE</td>
<td>ADD</td>
<td>ADE</td>
<td>SSC</td>
<td>CSE</td>
<td>ASC</td>
<td>CAE</td>
</tr>
<tr>
<td>07</td>
<td>STF</td>
<td>SIF</td>
<td>SEQ</td>
<td>SIA</td>
<td>STU</td>
<td>SAM</td>
<td>SAL</td>
<td>SEX</td>
</tr>
<tr>
<td>10</td>
<td>CSB</td>
<td>CST</td>
<td>CAD</td>
<td>CAT</td>
<td>NOT</td>
<td>AND</td>
<td>LOR</td>
<td>BLS</td>
</tr>
<tr>
<td>11</td>
<td>MPY</td>
<td>DIV</td>
<td>NDV</td>
<td>VID</td>
<td>STR</td>
<td>XCH</td>
<td>STC</td>
<td>STN</td>
</tr>
<tr>
<td>12</td>
<td>STF</td>
<td>SIF</td>
<td>SEQ</td>
<td>SIA</td>
<td>STU</td>
<td>SAM</td>
<td>SAL</td>
<td>SEX</td>
</tr>
<tr>
<td>13</td>
<td>LAL</td>
<td>DAV</td>
<td>SRM</td>
<td>LRS</td>
<td>SRS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** All unassigned order are illegal, namely the blanks in this table and orders whose first digits are 00, 01; 15, 16, or 17.

* Order is interrupted unless interrupt is disabled.
3.5.5 Table 5. Additional Mnemonics

CAN
CSN
ADN
CNN
CRN
ANN
ORN
EON
EQN
NAN
NON

are equivalent to

CAM
CSM
ADM
CNM
CRM
ANM
ORM
EOM
EQM
NAM
NOM

with an odd C field.

TEI  Transfer and enable interrupt is JDC  0
TRA  Transfer is JDC  1
TZP  Transfer if zero or plus is JDC  2
TN   Transfer if negative is JDC  3
TU   Transfer if unzero is JDC  4
TZ   Transfer if zero is JDC  5
TP   Transfer if plus is JDC  6
TZN  Transfer if zero or negative is JDC  7
TO   Transfer if overflow is JDC  8
TNO  Transfer if no overflow is JDC  9
TOR  Transfer if overflow and reset is JDC 10
TNOR Transfer if no overflow and reset is JDC 11
TLP  Transfer if logical plus is JDC 14
TLN  Transfer if logical minus is JDC 15

CALL is assembled as JSB3, and in addition it loads the subroutine into memory (see Chapter 4) and it fills up the current word so that the subroutine return can be made with a JLH 3,0.

CAJ is assembled as either CJU, CJF or CJS.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>4.1 Introduction</th>
<th>Change</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2 Card Format</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.2.1 Location Field</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.2.2 Mnemonic Field</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.2.3 Address Fields</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.2.4 Comments Field</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.2.5 Identification Field</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.3 Address Construction</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.3.1 Machine Evaluation of Address Expressions</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.3.2 Illegal Use of Names in Address Fields</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.4 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.1 Type 1 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.2 Type 2 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.3 Type 3 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.4 Type 4 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.5 Type 5 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.6 Type 6 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.7 Type 7 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.8 Type 7A Orders</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.4.9 Type 8 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.10 Type 9 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.11 Type 10 Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.4.12 Type 11 Orders</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.4.13 Type 12 Order</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.4.14 Illegal Orders</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.5 Pseudo Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.5.1 Directives</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.5.2 Data-Loading Pseudo Orders</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.6 Input/Output Pseudo Instructions</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.7 Macro Orders</td>
<td>3/5/63</td>
<td></td>
</tr>
<tr>
<td>4.8 Notes on Simple Programming in Assembler Language</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.9 Program Listing</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.10 Tables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.10.1 Table 1. Order Code Index</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.10.2 Table 2. Pseudo Orders</td>
<td>7/9/64</td>
<td></td>
</tr>
<tr>
<td>4.10.3 Table 3. BCD Tape and Card Code</td>
<td>11/8/63</td>
<td></td>
</tr>
</tbody>
</table>
4. NICAP, THE ASSEMBLY PROGRAM

4.1 Introduction

The assembly program is designed to allow the programmer writing in machine language to program without thought to the many different address constructions that are used internally and yet enable him to produce an efficient program. It is not intended that this complex assembler should replace compilers, but it is hoped that some jobs for which the programmer turns to a compiler because of involved addressing can now be handled by this assembler, giving a more efficient object program.

For this reason a very general format is allowed in the address field of most orders. This format is, in most cases, self-explanatory. The address field can contain, for example, a direct indication of multiple indexing, which will result in more than one order being assembled. In this sense, the assembler performs a compilation on the address field.

Additionally, to give the flexibility necessary to those who wish to write in a (1-1) transformation of the machine language, multiple field address formats representing each of the three (B, C and N) address fields are allowed.

To allow for future machine and system expansion, programs should be written in a relocatable form. This can be achieved simply by never using absolute addresses to refer to memory. If the first ORG pseudo-operation is omitted, then the program will be automatically relocated to start in the first free area of memory.
### 4.2 Card Format

<table>
<thead>
<tr>
<th>Location</th>
<th>Mnemonic</th>
<th>Address</th>
<th>Comment</th>
<th>Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>78</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>32</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>72</td>
<td>73</td>
</tr>
</tbody>
</table>
4.2.1 Location Field

Columns 1 to 6 are the location field and may contain a one to six-character name. A name may consist only of alphanumeric characters, and must contain at least one alphabetic character. A name is a symbolic representation of one of four elements in the machine, and it is given a title accordingly.

The four subtypes of names are:

1. **Symbol** the name of the location of a full word.
2. **Label** the name of the location of a quarter word.
3. **Tag** the name of a modifier register or index register.
4. **Register** the name of a fast register.

As with most assemblers, all names used must be defined at some point in the program. They are normally defined by appearing in the location field of a card. This will define both the type of name and its absolute value. An exception is made in the case of 24 names which are predefined.

These are M0, M1, ..., M15, and F0, F1, ..., F7 which are names for the modifiers 0 to 15 and the fast registers 0 to 7 respectively. These may not be additionally defined by the programmer.

Any name starting with SYS should be avoided by the programmer since all system program routine names will start thus. The type of name that is defined by a card is determined by the mnemonic field discussed below. Following the location field column 7 is blank to provide a separation between the name and the mnemonic fields.

An * in column 1 indicates a comment card. The remainder of the card is ignored.
4.2.2 Mnemonic Field

Columns 8 to 13 contain a one to six-character mnemonic. This mnemonic may be either:

(1) an order
(2) a pseudo order
(3) an I/O pseudo instruction

or

(4) a macro order

As a general rule, names appearing in the location fields of these types are defined as:

(1) labels
(2) symbols
(3) symbols

or

(4) labels

respectively.

Exceptions occur in the case of pseudo-orders (case (2)) and are noted in their descriptions below. Details of the operation of orders are given in Chapter 3, of the pseudo-orders and macro-orders in this chapter and of the operation of the I/O pseudo-instructions in Chapter 5. The address constructions of all mnemonics are listed later in this chapter.

Following the mnemonic field, Column 14 must contain a blank to separate the mnemonic field from the address field.
4.2.3 Address Fields

Columns 15 on to the first blank character after column 15 or on to column 72, whichever occurs first, contain the address information. This field can contain an arbitrary number of characters up to 58 which determine the address of the order. The various address constructions are listed in section 4.3.
4.2.4 Comments Field

Anything that occurs after the first blank following column 15 but not before column 33 or after column 72 is comment, and is simply reproduced on the output listing.

Comments may also be placed anywhere on a card with an * in column 1.
4.2.5 **Identification Field**

Columns 73 to 80 are also reproduced on the output listing, but are normally used for card identification only. They do not affect the program in any way.
4.3 Address Construction

The types of address format allowed depend, in part, on the mnemonic. The most general form of address that is allowed can be stated as being either:

(1) A register name $F$, e.g., $F3$ or some other name for it.

or

(2) Any algebraically meaningful expression $E$ containing numbers (decimally represented), symbols (representing numbers), tags (representing modifiers), the algebraic operators $+, -, *,$ (multiply) and $/$ (divide), and the parentheses $(, and $)$. Algebraically meaningful means that the expression satisfies the conventional rules and that multiplication is always written explicitly as $*$, e.g., $(10+A) * (M5+7)$ may not be written as $(10+A)(M5+7)$.

Various restrictions are applied to these rules for different classes of mnemonics; for example, most pseudo-orders do not allow the use of tags since pseudo-orders generally are "obeyed" at assembly time and tags, by definition, only have meaning at execution time.
4.3.1 Machine Evaluation of Address Expressions

Rules that are observed by the machine in calculating these addresses are as follows:

(1) If no tags are involved, the expression is evaluated modulo 8192 at assembly time.

(2) If tags are involved, the rules are more complex. Essentially an effort is made to write a piece of program that will construct the address at execution time with a minimum of orders. The result of this is that if any modifier appears inside a parenthesis or is involved in a multiplication or division, MO, the accumulator, FO and FI are changed before the order is executed.

Two problems can arise in complex expressions due to the fact that addresses are computed in the accumulator when modifiers are involved in multiplication, division, or parenthetical expressions. The first is that the accumulator will lose least significant bits if the result gets larger than $2^{14}$. Truncation modulo 8192 does not take place until the last item has been evaluated in the accumulator. The second problem concerns division which is performed in rounded floating-point to 44 places. The result is not truncated to an integer until the last evaluation in the accumulator has been completed. Thus the address field of

$$\frac{7}{2} + \frac{9}{2}$$

will give an address of

$$3 + 4 = 7,$$

whereas

$$\frac{M3}{2} + \frac{M4}{2}$$

will give an address of
\[ 3 \frac{1}{2} + 4 \frac{1}{2} = 8 \]

if M3 contains 7 and M4 contains 9 at execute time.

Division is not generally a useful operation, so it is best to avoid it unless either

(1) It does not involve tags.

(2) The answer is known to be an integer

or

(3) Only one division is used, and the result is not involved in a subsequence multiplication.

**Examples of Addressing**

(1) CAD M4+A will clear and add the number in location A plus the contents of M4 at execution time.

(2) If a matrix \( A_{ij} \) is stored by row in A to \( A + NM - 1 \) where M is the length of a row and N the length of a column, and tags I and J are the modifiers which contain i and j, then we can load modifier 4 with the address of \( A_{ij} \) with

\[ \text{CAM } 4, A + I-1 + M*(J-1) \]

where M is assumed to be defined as a symbol equal to the numerical value of M.
4.3.2 Illegal Use of Names in Address Fields

The use of symbols for labels and labels for symbols is permissible. It will cause an error to be listed, but the substitution will be made in a natural way, that is, 15-bit labels will be truncated to 13-bit symbols and 13-bit symbols will have two zero bits added to make them labels.

If registers or tags are used illegally, an error will be listed, and the name will now be interpreted as a symbol with value zero in order to allow the assembler to search for further errors.

Undefined names will be treated similarly.

Labels may be defined absolutely or relative to the program. Symbols may be so defined, and, additionally, may be defined relative to the common area or the erasable area. The address of an order must not be relocated more than once, or an error will be listed. There are, however, cases where this is legitimate. For example: CAM 8, A-B+C where A, B and C are program relocatable gives the address C which is relocatable plus the difference between A and B which is absolute. To handle this, the relocation bits are "exclusive ORed" so that double relocation (CAM 8, A-B) causes no relocation, etc.

Addresses that are too garbled for the assembler to understand cause the whole card to be rejected. Instead the quarter word 17700 is assembled. This is an illegal order which causes a hang-up if interrupt is disabled, and an interrupt otherwise.
4.4 Orders

Any name defined in the location field of an order is a label with value equal to the quarter-word address of the first control group formed by the order on that card.

The address construction for the order depends on the order type. The cases are listed separately below.

There is one form, called the normal form which can be used for all orders except for some extended mnemonics. The B, C and N fields are listed separately, in that order, and separated by commas in the normal address construction. The length of the order is determined by the B and C digits and the order type. The B field can be numeric between 0 and 15, and, in some cases, may be a tag or a register. The C field must be numeric between 0 and 3. The N field is a general address field which may be subject to restrictions for some order types. Other address constructions have been included in order that the programmer will not have to do an unnecessary amount of writing or remember exactly how each order forms its address. For example, to load the number from location A into the accumulator, the order

\[ \text{CAD } 8,3,A \]

can be written in the normal form. It can, however, also be written as

\[ \text{CAD } A \]
4.4.1 Type 1 Orders

ADD  Add
AND  AND
CAD  Clear Add
CAT  Clear Add Twice
CSB  Clear Subtract
CST  Clear Subtract Twice
DAV  Difference Absolute Values
DIV  Divide
LAL  Load A Least
LOR  Logical OR
MPY  Multiply
NDV  Negative Divide
NOT  NOT
SUB  Subtract
VID  Inverse Divide

The straightforward way to use these instructions is to use a single address field. If this consists of a register name \( F \), say \( F_5 \), then the order uses the contents of that fast register as the operand. It is assembled as one short order, e.g., \( \text{CAD} F_5 \) has a \( B \) field of 5 and a \( C \) field of 3, so is equivalent to \( \text{CAD} 5,3 \), in the normal form and puts the contents of \( F_5 \) in the accumulator. If the address field consists of an expression \( E \), the value of the address at execution time is the value of the expression, using the value of the modifiers current at execution time. This may result in more than one order being compiled. For example the following pairs are equivalent:

- \( \text{CAD} M_5 \) is equivalent to \( \text{CAD} 5,0 \)
- \( \text{CAD} M_5 + 301 \) is equivalent to \( \text{CAD} 5,2,301 \)
- \( \text{CAD} M_4 + M_7 \) is equivalent to \( \text{ATN} 4,0 \),
  \( \text{CAD} 7,0 \)

and

Date: 3/5/63
Section: 4.4.1
Page: 1 of 2
Change:
CAD M4-M7+301-M9 is equivalent to ATN 7,0
SFN 9,0
CAD 4,2,301

These instructions fill the accumulator with the number from the memory address indicated.

Thus multiple indexing, which on the machine is performed by preceding the instruction by a series of "to next"-type instructions, can be indicated in the address field.

Another form of addressing for this class of orders is the "normal" form

B, C, E

where B is a number, C is a number and E is any expression which should be blank if the order is short, that is if

\[ C = 0,1 \] or if \[ C = 3 \text{ and } B \neq 8, 9, \text{ or } 10 \]

B may also be a tag if \( C \neq 3 \), or a register if \( C = 3 \).

The third form consists of any expression E followed by the decimal point (.). This is equivalent to \( B = 9 \) and \( C = 3 \), so that the address is used as an integer operand.

Thus ADD E. is equivalent to ADD 9,3,E, e.g.,

ADD M5+7.

adds the integer 7 plus the contents of modifier 5 taken as an integer into the accumulator. Note here that the top bit of the 13-bit number in the modifier is used as a two's complement sign bit. Thus 8191 is equivalent to -1, 8190 to -2, ..., 4096 is equivalent to -4096 but 4095 is +4095.
### 4.4.2 Type 2 Orders

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASC</td>
<td>ADD to Store and Clear</td>
</tr>
<tr>
<td>SAL</td>
<td>Store A Least</td>
</tr>
<tr>
<td>SAM</td>
<td>Store A Most</td>
</tr>
<tr>
<td>SEQ</td>
<td>Store with Exponent Equal</td>
</tr>
<tr>
<td>SIF</td>
<td>Store Integer Part in Floating Point</td>
</tr>
<tr>
<td>SRM</td>
<td>Store Remainder</td>
</tr>
<tr>
<td>SSC</td>
<td>Subtract, Store and Clear</td>
</tr>
<tr>
<td>STC</td>
<td>Store and Clear</td>
</tr>
<tr>
<td>STF</td>
<td>Store Fixed Point Rounded</td>
</tr>
<tr>
<td>STN</td>
<td>Store Negatively</td>
</tr>
<tr>
<td>STR</td>
<td>Store Rounded and Normalized</td>
</tr>
<tr>
<td>STU</td>
<td>Store Unnormalized but Rounded</td>
</tr>
<tr>
<td>XCH</td>
<td>Exchange</td>
</tr>
</tbody>
</table>

These orders can have address field identical to type 1 orders except that if the order would finally assemble with $C = 3$ and $B = 1$ or $B \geq 9$ it is illegal. That is, Fl may not be used, the decimal point may not be used and there additionally is a restriction on $B$ if the normal address structure is used with $C = 3$. 
4.4.3 Type 3 Orders

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADE</td>
<td>Add to Exponent</td>
</tr>
<tr>
<td>ATN</td>
<td>Add to Next</td>
</tr>
<tr>
<td>*BBF</td>
<td>Busy Block Flipflop</td>
</tr>
<tr>
<td>CAE</td>
<td>Clear Add Exponent</td>
</tr>
<tr>
<td>CSE</td>
<td>Clear Subtract Exponent</td>
</tr>
<tr>
<td>*FIF</td>
<td>Free Block Flipflop</td>
</tr>
<tr>
<td>*IBT</td>
<td>Initiate Block Transfer</td>
</tr>
<tr>
<td>JIH</td>
<td>Jump to Left-hand Side</td>
</tr>
<tr>
<td>LIN</td>
<td>Load In Register</td>
</tr>
<tr>
<td>LRS</td>
<td>Long Right Shift</td>
</tr>
<tr>
<td>ORB</td>
<td>OR to B Digits of Next</td>
</tr>
<tr>
<td>*PID</td>
<td>Prepare Input Device</td>
</tr>
<tr>
<td>*POD</td>
<td>Prepare Output Device</td>
</tr>
<tr>
<td>SBE</td>
<td>Subtract from Exponent</td>
</tr>
<tr>
<td>SFN</td>
<td>Subtract from Next</td>
</tr>
<tr>
<td>SRS</td>
<td>Short Right Shift</td>
</tr>
</tbody>
</table>

The address fields for these orders are identical to those of type 1 orders, except that the orders must not assemble with \(C = 3\) and \(B \neq 8\). Therefore, the decimal point may not be used, and a fast register may not appear in the address field. Note that for these orders, the address is generally the operand.

* These orders cause an interrupt and should not be used when operating within the system.
**4.4.4 Type 4 Orders**

- ADM: Add to Modifier
- ANM: AND to Modifier
- CAM: Clear and Add to Modifier
- CNM: Clear and Negate to Modifier
- CRM: Circular Rotate Modifier Right
- CSM: Clear and Subtract from Modifier
- EOM: Exclusive OR to Modifier
- EQM: Equivalence to Modifier
- NAM: Negate and AND with Modifier
- NOM: Negate and OR with Modifier
- ORM: OR with Modifier
- SBM: Subtract from Modifier

In addition to the normal address construction B, C, E two formats are allowed for this type of order.

For short orders with no address, the modifier alone can be written. E.g.,

CAM B where B is numeric (< 16) or is a tag.

This clears modifier B unless modified by a previous "to next" instruction.

The second format is

CAM B,E

This order will be made short or long as E does or does not involve a numeric quantity. E.g.,

CAM 5,M7

assembles as

---

Date: 3/5/63
Section: 4.4.4
Page: 1 of 2
Change:
ATN 7,0,
CAM 5,0,

while

CAM 5, M7+3

assembles as

ATN 7,0,
CAM 5,2,3

The first address (B) is the modifier referred to by the instruction, the second address field is the operand. Thus

ADM 3, M7 + 3

adds Modifier 7 and the integer 3 to modifier 3.

Under no circumstances may B be a register name.
4.4.5 **Type 5 Orders**

These are the "to next" modification of the preceding group with the C field equal to 1 or 3 instead of 0 or 2. They can be obtained from the type 4 orders by replacing the final M with an N, e.g., ADM becomes ADN.

They are:

- **ADN**
  - Add to Next
- **ANN**
  - AND with Modifier and Add to Next
- **CAN**
  - Add Address to Next
- **CNN**
  - Clear and Negate to Next
- **CRN**
  - Circulate Rotate and Add to Next
- **CSN**
  - Subtract Address from Next
- **EON**
  - Exclusive OR with Modifier, and Add to Next
- **EQN**
  - Equivalence with Modifier, and Add to Next
- **NAN**
  - Negate and AND with Modifier and Add to Next
- **NON**
  - Negate and OR with Modifier and Add to Next
- **ORN**
  - OR with Modifier and Add to Next
- **SBN**
  - Subtract from Next

The address field of a type 5 order can have the same format as type 4 order except that the normal address construction with \( C = 0 \) or 2 may not be used.

---

\( ^+ \) These operations perform no operation that cannot also be achieved by ATN or SFN except that CAN or CSN can be used as a short no operation provided that they are not preceded directly by an ORB order.
In the following group, orders which call for "Count" mean add one to the indicated modifier. The Jump occurs if the modifier is nonzero, except for C J Z.

4.4.6 Type 6 Orders

C J F  Count and Jump to First (if nonzero)
C J S  Count and Jump to Second (if nonzero)

The normal form of addressing

C J S  B, C,

may be used. The second comma may be omitted to get

C J S  B, C

C must be numeric (0 to 3), B may be numeric or it may be a tag.

These orders would not usually be used; rather the C A J (type 7A) order would be used unless the user is interested in optimizing a very short piece of program to make use of a fast loop in F8 and F9. (See Chapter 3 for details of the orders.)
4.4.7 Type 7 Orders

CJU Count and Jump if Unzero
CJZ Count and Jump if Zero
JDC Jump on Diversity of Conditions
JNM Jump if Negative Modifier
JPM Jump if Positive Modifier
JSB Jump to Subroutine
JUM Jump if Unzero Modifier
JZM Jump if Zero Modifier

In the normal form

CJU B,C,E

B may only be a number or a tag. This address construction should normally be avoided, since it is usually better to refer to locations of orders by labels, which represent 15-bit rather than 13-bit addresses. (The extra two bits are the quarter-word address 0 to 3.)

This construction would find use in branching to a table of words, e.g.,

JPM5,1,A+M7

would jump on positive M5 to the second quarter word of A plus Modifier 7 (if A is a symbol). Library subroutines will also make use of this construction so that only one label is used in the entire subroutine, e.g., in the COSINE routine, we might find constructions

CJU4,2,C0S+7

to jump to the \((4\times7+1) = 29\)th quarter word after the start of the subroutine COS. In fact this will work even if the subroutine were not to
start on a word boundary since the following rule is obeyed for this order
type and for types 7A and 8:

If the first element in the N field expression is a label, the
quarter-word part of it (two bits) is added to the C field. The bottom
two bits of the answer are retained in the C field, and the carry is
added to the word address equivalent of the label, which is then truncated
to a symbol for use in evaluating the expression, e.g., if COS is location
100, quarter word 3,

\[ \text{CJU4,2,COS+7} \]

is equivalent to

\[ \text{CJU4,1,108} \]

However, \textbf{beware:}

\[ \text{CJU4,2,7+COS} \]

is equivalent to

\[ \text{CJU4,2,107} \]

If the latter of these constructions is used, a possible error
pointer will be given in the output listing.

If the C required is zero, the field and one of the commas may
be omitted. Thus:

\[ \text{JSB3,COS} \]

will jump to the quarter-word in which the COS subroutine starts.

\textbf{NOTE:} For library programs which always return to the left-hand side
of a word, it is better to use the pseudo-operation CALL instead
of JSB3, (see below).
Type 7A Order

CAJ  Count and Jump if Nonzero

This order has the same address construction as type 7 orders; it will assemble as either CJU, CJF or CJS according to the range and position of the jump. However, it will not necessarily make the most efficient decision, so, in important, frequently-used short loops, it is wiser to hand tailor it with CJF or CJS.
4.4.8 Type 8 Orders

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEI</td>
<td>Transfer and Enable Interrupt</td>
</tr>
<tr>
<td>TLN</td>
<td>Transfer if Logical Negative</td>
</tr>
<tr>
<td>TLP</td>
<td>Transfer if Logical Positive</td>
</tr>
<tr>
<td>TN</td>
<td>Transfer if Negative but Not Zero</td>
</tr>
<tr>
<td>TNO</td>
<td>Transfer if No Overflow</td>
</tr>
<tr>
<td>TNOR</td>
<td>Transfer if No Overflow and Reset Overflow</td>
</tr>
<tr>
<td>TO</td>
<td>Transfer if Overflow</td>
</tr>
<tr>
<td>TOR</td>
<td>Transfer if Overflow and Reset Overflow</td>
</tr>
<tr>
<td>TP</td>
<td>Transfer if Positive but Not Zero</td>
</tr>
<tr>
<td>TRA</td>
<td>Transfer</td>
</tr>
<tr>
<td>TU</td>
<td>Transfer if Unzero</td>
</tr>
<tr>
<td>TZ</td>
<td>Transfer if Zero</td>
</tr>
<tr>
<td>TZN</td>
<td>Transfer if Zero or Negative</td>
</tr>
<tr>
<td>TZP</td>
<td>Transfer if Zero or Positive</td>
</tr>
</tbody>
</table>

These are similar to type 7 orders except that they do not require a B field and therefore cannot use the normal address form. The address constructions are those of type 7 orders with the B field and first comma omitted, e.g.,

```
TRA START to transfer unconditionally to the order labelled START
```

and

```
TZ 2,COSI+7 to transfer to the 30th quarter word after the start of the COSINE subroutine if the accumulator is zero
```
4.4.9 Type 9 Orders

LFR  Load Fast Register
SFR  Store Fast Register

With normal address construction the B field must be either numeric between 2 and 7 or a register name excluding FO.

The order is long if C = 2 or 3 and short otherwise. The C field and the following comma may be omitted, in which case the order is made long if the N field contains a numerical quantity, e.g.,

LFR5,M9+7

assembles as

ATN9,0,
LFR5,2,7

whereas

LFR5,ML3

assembles as

ATNL3,0,
LFR5,0,

NOTE: SFR may not use Fl.
4.4.10 **Type 10 Orders**

LDM Load Modifier

This order is always long so the C digits have no meaning in normal address construction. Therefore the C field and the preceding comma may be omitted. If the N address field is zero, it and the preceding comma may be omitted.

The B address field must not be a register.
4.4.11 Type 11 Orders

SIA Store Integer in Address
SEX Store Exponent

These orders are short always and C has no meaning. B must not be a register. If the normal address format is used, the N field should be blank. Everything except for the B field may be omitted.
Type 12 Orders

BLS     Binary Left Shift
or     LFl     Load Fast Register 1

In all cases this instruction loads Fl with an operand. In addition it performs a logical single binary left shift unless C = 3 and B < 8 and B ≠ 1. If no address is used in BLS, it assembles as BLS L3. Otherwise it has the construction of type 3 orders. LFl may only use a number 0, and 2-7 or a fast register name as an address. It always assembles with an N,3, case.
4.4.13 Type 13 Orders

*ASN   Add Special Register to Next
*SSN   Subtract Special Register from Next
*SSR   Store in Special Register
*HLT   Halt

These orders cause an interrupt, so should not be used when using the system.

Normal address construction can be used, but B and C must be numeric and the N field must be blank. Alternatively, one expression field only can be used. It must include no tags or fast registers. The numerical value is used modulo 64 in the B and C bit positions.
4.4.14  Illegal Orders

Mnemonics which cannot be understood, or those with addressing sufficiently garbled are assembled as the quarter word 17700 which is an illegal order.
4.5 Pseudo Orders

Pseudo orders fall into two categories:

a) Directives to the assembler which cause no words to be assembled in the object program, but usually either determine the memory location of subsequent orders, make an entry in the name table, or do both.

b) Indications to the assembler that what follows is to be used as data.

The address field of either group must be computable at assembly time, that is, they may not contain modifiers.
4.5.1 **Directives**

During the assembly phase, the assembler reads the cards, one by one, assembles each card into one or more 13-bit groups, and assigns them to consecutive control groups or 13-bit locations. To do this, the assembly has a "location counter" which consists of a 13-bit word counter W, and a 2-bit quarter-word counter Q. It is incremented by one quarter for each control group assembled. This can be modified by the following groups of directives. It is initially set to the number of transfer vectors to be generated by the program.

**ORG (Origin)** The address field of this pseudo order is put in the word counter W and the quarter word counter Q is cleared to zero. Consequently the next order assembled goes into the start of location W. Generally, there is no need to use an ORG card; the program will automatically be placed at the beginning of the available memory.

**FIL (Fill)** This may have a numeric address between 0 and 3. The zero may be omitted. Its action is to assemble the order CAM 0,1 as many times as necessary to make Q equal to the address in the FIL. CAM 0,1 acts as a no operation except after an ORB instruction. The effect of FIL 0 for example, is to advance the instruction counter to the next word boundary unless it was already on a word boundary.

**FLD (Fill Double)** This may have a numeric address between 0 and 7. It is similar to FIL except that it takes note of the oddness or evenness of the word counter W. It assembles CAM 0,1 instructions until 4 times (the bottom bit of W) + Q is equal to the address in the FLD. Thus

\[
\text{FLD } 4
\]

advances the instruction counter to the next odd word boundary, while

\[
\text{FLD}
\]

advances it to the next even word boundary.
If any of the above pseudo orders have a name in the location field, it is set as a symbol having the new value of the word counter W.

BSS (Block Started by Symbol)  First an FIL O is performed, and then the block of locations whose length is specified by the address field is reserved, that is, the word counter W is increased by that number. The name in the location field is made a symbol equal to the first location of the block reserved.

BES (Block Ended by Symbol)  Similar to BSS, except that the symbol defined in the location field is equated to the word address immediately following the last word reserved.

ASSIGN  This performs a FIL. It would normally have nothing in the location field, but if it did, the name would be made a symbol equal to the current word address after the FIL. The address field of the ASSIGN can only contain a sequence of names not defined elsewhere, each followed by a comma, except for the last. They are entered in the name table as symbols, each assigned a value of a consecutive word location. The locations are reserved, that is, the word counter W is incremented by a number equal to the number of symbols defined, e.g.,

\[ \text{ASSIGN } X, A1, 23K \]

defines three new symbols X, A1, and 23K and reserved one word for each.

GO  This pseudo order signals the end of the program. The address field may only contain a label which will be the address of the first order to be obeyed. If no ORG was used, and the only pseudo orders preceding the first order to be obeyed are EQU's, then this can have a blank address field.
The following six pseudo orders do not affect the instruction counter.

**EQU3 (Equate to Symbol)** The name in the location field is defined as a symbol with the value given in the address field, e.g.,

\[ \text{AB EQU A+B} \]

defines the symbol AB as having a value equal to the sum of the values of the two symbols A and B.

**EQU4 (Equate to Label)** This defines the location field name as a label. The address of the pseudo order must be numeric or another label.

**EQUF (Equate to Modifier)** The location field name is set as a tag with the value given in the address field which must be numeric or another tag.

**EQUF (Equate to Fast Register)** The location field name is set as a register with the value given in the address field which must be numeric or another register.

**MACRO** is followed by a string of dummy names (for example MACRO X,Y,Z) each followed by a comma except for the last. The contents of the location field of this pseudo order do not define a name; they define a macro operation.

This pseudo operation is followed by a string of machine operations terminated by the pseudo operation END. Each time the macro name defined by this MACRO appears, this string of instructions is copied in. The dummy symbols, labels, tags or registers X, Y, Z, ..., W are used in the addresses of the instructions defining a macro. When the macro is used, these addresses must be defined in an identical format.

Date: 7/9/64
Section: 4.5.1
Page: 3 of 5
Change: 2
Example:

```
CRASH MACRO A, B, C
CAD A
MPY B
STR C
END
```

would not define symbols A, B and C, that is, they would not be entered in
the name table. When the macro instruction CRASH 10, ALPHA, 11 is used,
the machine instructions

```
CAD 10
MPY ALPHA
STR 11
```

are assembled.

Macro definitions may use pseudo orders except for the EQU and
ORG types.

END  This pseudo order terminates a macro definition as described above.

COMMON  This must have a numeric address N. It causes the next N words
of the COMMON area to be set aside for the symbol in the location field.
It is thus similar to the BSS instruction in the common area.

Example:

```
A COMMON 10
B COMMON 13
C COMMON 21
```

would allocate 44 words of COMMON. A would be 0, B 10 and C 23 relative
to this area.
ERASE   This controls the erasable area exactly as COMMON controls the common area.

ENTRY   This must be followed by one or more defined names, separated by commas. These are the names by which the program segment being assembled may be CALLed by other programs.
4.5.2 Data-Loading Pseudo Orders

DECQ can be followed by a sequence of addresses separated by commas. Each assembles into one control group, e.g.,

DECQ 7, A+19,3

forms the three quarter words

7, A+19 and 3.

The addresses can be any expressions involving numbers and symbols.

OCTQ is identical to DECQ except that numbers are converted base 8, e.g.,

OCTQ 15071, 32

assembles the two quarter words

\[
\begin{array}{cccccc}
1 & 101 & 000 & 011 & 001 \\
1 & 5 & 0 & 7 & 1 & 3 & 2
\end{array}
\]

OCTQ and DECQ cause outside names to be defined as labels.

CHR (Character) This pseudo order is followed by one decimally represented address N followed by a comma, then the following N characters are packed, eight per word, into the next N/8 words. The last word is filled up with blank characters once it is started. This is the only card for which the address field does not terminate at the first blank after column 15.

Before the words are assembled, a FIL is performed, and then any name in the location field is equated to a symbol.
with the value of the current word counter. The character code used is the standard IBM BCD tape code given in Table 3 at the end of this chapter. Two six-bit characters are packed in adjacent six-bit groups in the least significant 12 bits of each control group. Thus one word has the following format:

\[
\begin{array}{cccccccccc}
0 & c1 & c2 & 0 & c3 & c4 & 0 & c5 & c6 & 0 & c7 & c8 \\
\end{array}
\]

First performs a FIL, then equates the name as a symbol equal to the current word counter. Decimal numbers may appear in the address field separated from one another by commas. Each is converted into a full word floating-point number. The number may be punched with or without a decimal point (no point is identical to putting the point last) and with or without a decimal exponent. An exponent must be preceded either by E, a sign, or e and a sign. The number should lie between \(10^{-38}\), but can contain an arbitrary number of digits, although only 13 digits (approximately) are retained.

Example of numbers:

<table>
<thead>
<tr>
<th>Punched as</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>72</td>
<td>(+.74 \times 10^2)</td>
</tr>
<tr>
<td>-27.1</td>
<td>(-.271 \times 10^2)</td>
</tr>
<tr>
<td>+30.E-02</td>
<td>(+.301 \times 10^1)</td>
</tr>
<tr>
<td>7.5E06</td>
<td>(+.75 \times 10^7)</td>
</tr>
<tr>
<td>-3.32-05</td>
<td>(-.332 \times 10^{-4})</td>
</tr>
<tr>
<td>-3.32E-5</td>
<td>(-.332 \times 10^{-4})</td>
</tr>
</tbody>
</table>

The above pseudo orders should not cause more than seven full words or 31 quarter words to be assembled.
CALL is identical to writing

    JSB 3,
    FIL

where the address field of the CALL follows the comma in the JSB instruction. It may only contain a name. If this name is not defined in the program, a transfer vector TRA NAME will be assembled at the front of the program.
4.6 Input/Output Pseudo Instructions

These are a means of writing subroutine control words, usually for Input/Output subroutines. Each mnemonic first performs a FIL, then equates the location field name to the current word counter as a symbol. Address fields are assembled into the appropriate control groups of one full word and any necessary control bits are set in that word. The instructions allowed are described in Chapter 5. Note that the address fields must be computable at assembly time.
4.7 **Macro Orders**

The address field of a macro order may contain a series of expressions or register names, separated by commas. These are assigned to the dummy symbols in the macro definition as shown in the example in section 4.5.1.

Restrictions on the address fields of macro orders are precisely those due to their use within the macro definition.
4.8 Notes on Simple Programming in Assembler Language

To write simple programs, it is not necessary to make use of many of the different address formats allowed, or to learn many rules.

With arithmetic orders, the address need only be either a fast register, e.g.,

```
ADD F3
```

or a memory location defined by an expression, e.g.,

```
CAD 9+M5.
```

Names used in these address fields are usually symbols defined in BSS, BES or ASSIGN pseudo operations.

Modifier register orders naturally require an indication of the modifier also, so this comes first followed by a comma, e.g.,

```
ADM 5,3+M7
```

"add to modifier 5, three plus modifier 7."

Jump or transfer orders must give the address of another order. This address is usually a label, e.g.,

```
CJU M5,AA
```

"Count and jump if unzero modifier 5 to the order labeled AA."

It is necessary to go to other formats only to gain speed in important places.

Example 1. Polynomial Evaluation.

Suppose we wish to evaluate a polynomial \( p(x) = A_N + A_{N-1}x + \ldots + A_0x^N \) where the coefficients \( A_0, A_1, \ldots, A_N \) are in locations \( A, A+1, \ldots \) up to \( A + N - 1 \) and \( x \) is in location X.
The program:

CSM 4,N
CAD A
L1 MPY X
ADD A+N+1+M4
CAJ 4,L1

will do it. However, this will only be assembled as a short loop if L1 falls in the right place. This can be avoided by making sure that it does fall in the right place with a FIL.

Secondly, X is fetched from the memory on each pass. This can be avoided by putting it in a fast register.

Thirdly, the ADD instruction is long; it can be made short by using the C = 1 option. (In this case it makes no difference because the loop is already less than eight quarter words.)

After rewriting, the program is:

CSM 4,N
CAM 5,A+1
LPR 2,X
CAD A
FLD
L1 MPY F2
ADD 5,1,
CJF 4

If the FLD causes no CAM 0,1 instructions to be assembled, the program is, of course, that much faster.
Example 2.

To save time setting several modifiers, it is better to use LFR instructions.

Add the three N vectors A, B and C, each stored in consecutive locations in memory. Store the result starting at location D.

```
LFR 5,Sl          Load four modifiers
CSM 8,N
FIL
Ll    CAD 4,l,
    ADD 5,l,
    ADD 6,l,
    STR 7,l,
    CAJ 8,Ll
    ...
    ...
    ...
    ...
    ...
FIL
Sl    DECQ A,B,C,D  Constants for loop
```

Execution of a program should be terminated by a CALL SYSTEM or a CALL SYSERR in order not to obtain or to obtain a dump respectively.
After the assembly has been performed, a listing will be prepared giving the original program and its binary form side by side. The format, across a line is:

Card Number in Decimal
Location in Decimal
Location in Octal
Octal Code first quarter word
second quarter word
third quarter word
fourth quarter word

Source Language

If any errors or suspected errors are found, an * is printed after the card number.

Following the program is a list of all errors referenced to the card number on which they occurred, and a name table list.
### Table 1. Order Code Index

<table>
<thead>
<tr>
<th>Order</th>
<th>Type</th>
<th>Section No.</th>
<th>Page No.</th>
<th>Order</th>
<th>Type</th>
<th>Section No.</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
<td>CST</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>ADE</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>DAV</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>ADM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>DIV</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>ADN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>EOM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
</tr>
<tr>
<td>AND</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
<td>EON</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
</tr>
<tr>
<td>ANM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>EQM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
</tr>
<tr>
<td>ANN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>EQN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
</tr>
<tr>
<td>ASC</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>*FBE</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
</tr>
<tr>
<td>*ASN</td>
<td>13</td>
<td>4.4.13</td>
<td>1</td>
<td>*IBT</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
</tr>
<tr>
<td>ATN</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>JDC</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>*BBF</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>JLH</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
</tr>
<tr>
<td>BLS</td>
<td>12</td>
<td>4.4.12</td>
<td>1</td>
<td>JNM</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>CAD</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
<td>JPM</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>CAE</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>JSB</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>CAJ</td>
<td>7A</td>
<td>4.4.7</td>
<td>3</td>
<td>JUM</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>CAM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>JZM</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
</tr>
<tr>
<td>CAN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>LAL</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CAT</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
<td>LDM</td>
<td>10</td>
<td>4.4.10</td>
<td>1</td>
</tr>
<tr>
<td>CJF</td>
<td>6</td>
<td>4.4.6</td>
<td>1</td>
<td>LFI</td>
<td>12</td>
<td>4.4.12</td>
<td>1</td>
</tr>
<tr>
<td>CJS</td>
<td>6</td>
<td>4.4.6</td>
<td>1</td>
<td>LFR</td>
<td>9</td>
<td>4.4.9</td>
<td>1</td>
</tr>
<tr>
<td>CJU</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
<td>LIN</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
</tr>
<tr>
<td>CJZ</td>
<td>7</td>
<td>4.4.7</td>
<td>1</td>
<td>LOR</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CNM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>LRS</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
</tr>
<tr>
<td>CNN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>MPY</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CRM</td>
<td>1</td>
<td>4.4.4</td>
<td>1</td>
<td>NAM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
</tr>
<tr>
<td>CRN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>NAN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
</tr>
<tr>
<td>CSB</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
<td>NDV</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>CSE</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>NOM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
</tr>
<tr>
<td>CSM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>NON</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
</tr>
<tr>
<td>CSN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>NOT</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
</tbody>
</table>

Date: 7/9/64  
Section: 4.10.1  
Page: 1 of 2  
Change: 1
<table>
<thead>
<tr>
<th>Order</th>
<th>Type</th>
<th>Section No.</th>
<th>Page No.</th>
<th>Order</th>
<th>Type</th>
<th>Section No.</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORB</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>STF</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
</tr>
<tr>
<td>ORM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>STN</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
</tr>
<tr>
<td>ORN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>STR</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
</tr>
<tr>
<td>*FID</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>STU</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
</tr>
<tr>
<td>*FOD</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>SUB</td>
<td>2</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>SAL</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TEI</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SAM</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TLN</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SBE</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>TLP</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SBM</td>
<td>4</td>
<td>4.4.4</td>
<td>1</td>
<td>TN</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SBN</td>
<td>5</td>
<td>4.4.5</td>
<td>1</td>
<td>TNO</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SEQ</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TNOR</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SEX</td>
<td>11</td>
<td>4.4.11</td>
<td>1</td>
<td>TO</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SFN</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>TOR</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SFR</td>
<td>9</td>
<td>4.4.9</td>
<td>1</td>
<td>TP</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SIA</td>
<td>11</td>
<td>4.4.11</td>
<td>1</td>
<td>TRA</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SIF</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TU</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SRM</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TZ</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SRS</td>
<td>3</td>
<td>4.4.3</td>
<td>1</td>
<td>TZN</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>SSC</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td>TZP</td>
<td>8</td>
<td>4.4.8</td>
<td>1</td>
</tr>
<tr>
<td>*SSN</td>
<td>13</td>
<td>4.4.13</td>
<td>1</td>
<td>VID</td>
<td>1</td>
<td>4.4.1</td>
<td>1</td>
</tr>
<tr>
<td>*SSR</td>
<td>13</td>
<td>4.4.13</td>
<td>1</td>
<td>XCH</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
</tr>
<tr>
<td>STC</td>
<td>2</td>
<td>4.4.2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.10.2: Pseudo Orders

<table>
<thead>
<tr>
<th>Pseudo Order</th>
<th>Section No.</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASSIGN</td>
<td>4.5.1</td>
<td>2</td>
</tr>
<tr>
<td>ES</td>
<td>4.5.1</td>
<td>2</td>
</tr>
<tr>
<td>BSS</td>
<td>4.5.1</td>
<td>2</td>
</tr>
<tr>
<td>CALL</td>
<td>4.5.2</td>
<td>3</td>
</tr>
<tr>
<td>CHR</td>
<td>4.5.2</td>
<td>1</td>
</tr>
<tr>
<td>COMMON</td>
<td>4.5.1</td>
<td>4</td>
</tr>
<tr>
<td>DEC</td>
<td>4.5.2</td>
<td>2</td>
</tr>
<tr>
<td>DECQ</td>
<td>4.5.2</td>
<td>1</td>
</tr>
<tr>
<td>END</td>
<td>4.5.1</td>
<td>4</td>
</tr>
<tr>
<td>ENTRY</td>
<td>4.5.1</td>
<td>5</td>
</tr>
<tr>
<td>EQUIF</td>
<td>4.5.1</td>
<td>3</td>
</tr>
<tr>
<td>EQL</td>
<td>4.5.1</td>
<td>3</td>
</tr>
<tr>
<td>EQUM</td>
<td>4.5.1</td>
<td>3</td>
</tr>
<tr>
<td>EQUS</td>
<td>4.5.1</td>
<td>3</td>
</tr>
<tr>
<td>ERASE</td>
<td>4.5.1</td>
<td>5</td>
</tr>
<tr>
<td>FIL</td>
<td>4.5.1</td>
<td>1</td>
</tr>
<tr>
<td>FLD</td>
<td>4.5.1</td>
<td>1</td>
</tr>
<tr>
<td>GO</td>
<td>4.5.1</td>
<td>2</td>
</tr>
<tr>
<td>MACRO</td>
<td>4.5.1</td>
<td>3</td>
</tr>
<tr>
<td>OCTQ</td>
<td>4.5.2</td>
<td>1</td>
</tr>
<tr>
<td>ORG</td>
<td>4.5.1</td>
<td>1</td>
</tr>
</tbody>
</table>
### Table 3. BCD Tape and Card Code

**Table of Permissible Characters**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>blank</td>
<td>00</td>
<td>G</td>
<td>12 7</td>
<td>67</td>
<td>W</td>
<td>00 6</td>
<td>26</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>12</td>
<td>H</td>
<td>12 8</td>
<td>70</td>
<td>X</td>
<td>0 7</td>
<td>27</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>I</td>
<td>12 9</td>
<td>71</td>
<td>Y</td>
<td>0 8</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>02</td>
<td>J</td>
<td>11 1</td>
<td>41</td>
<td>Z</td>
<td>0 9</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>03</td>
<td>K</td>
<td>11 2</td>
<td>42</td>
<td></td>
<td>= 3-8 13</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>04</td>
<td>L</td>
<td>11 3</td>
<td>43</td>
<td>(</td>
<td>0 4-8 34</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>05</td>
<td>M</td>
<td>11 4</td>
<td>44</td>
<td>)</td>
<td>12 4-8 74</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>06</td>
<td>N</td>
<td>11 5</td>
<td>45</td>
<td>+</td>
<td>12 0</td>
<td>72</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>07</td>
<td>O</td>
<td>11 6</td>
<td>46</td>
<td>-</td>
<td>11 0</td>
<td>52</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
<td>P</td>
<td>11 7</td>
<td>47</td>
<td>'</td>
<td>4-8 14</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
<td>Q</td>
<td>11 8</td>
<td>50</td>
<td>+</td>
<td>12 60</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>12 1</td>
<td>61</td>
<td>R</td>
<td>11 9</td>
<td>51</td>
<td>-</td>
<td>11 40</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>12 2</td>
<td>62</td>
<td>S</td>
<td>0 22</td>
<td>*</td>
<td>11 4-8 54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12 3</td>
<td>63</td>
<td>T</td>
<td>0 23</td>
<td>/</td>
<td>0 1</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>12 4</td>
<td>64</td>
<td>U</td>
<td>0 24</td>
<td>$</td>
<td>11 3-8 53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>12 5</td>
<td>65</td>
<td>V</td>
<td>0 25</td>
<td>3-8 33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>12 6</td>
<td>66</td>
<td></td>
<td></td>
<td></td>
<td>12 3-8 73</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† Character is not normally used. When it is used it will not be considered a sign.
CHAPTER 5. SYSTEM INPUT/OUTPUT AND AUXILIARY STORAGE

TABLE OF CONTENTS

5.3 Input-Output with Conversion

5.3.1 Format Control
   5.3.1.1 Field Descriptions
   5.3.1.2 Multiple Field Descriptions
   5.3.1.3 Hollerith Fields
   5.3.1.4 Control Characters
   5.3.1.5 Relation of Format to the I/O List

5.3.2 Assembling the I/O List in NICAP
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Change</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Direct Communication with Back-Up Storage and I/O Tapes</td>
<td></td>
<td>7/10/64</td>
</tr>
<tr>
<td>5.2.1</td>
<td>SYSIØ</td>
<td>7/10/64</td>
<td></td>
</tr>
<tr>
<td>5.2.2</td>
<td>SYSAUX</td>
<td>7/10/64</td>
<td></td>
</tr>
<tr>
<td>5.3</td>
<td>Input/Output with Conversion</td>
<td></td>
<td>11/08/63</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Format Control</td>
<td></td>
<td>11/08/63</td>
</tr>
<tr>
<td>5.3.1.1</td>
<td>Field Descriptions</td>
<td>1</td>
<td>7/10/64</td>
</tr>
<tr>
<td>5.3.1.2</td>
<td>Multiple Field Descriptions</td>
<td></td>
<td>11/08/63</td>
</tr>
<tr>
<td>5.3.1.3</td>
<td>Hollerith Fields</td>
<td></td>
<td>11/08/63</td>
</tr>
<tr>
<td>5.3.1.4</td>
<td>Control Characters</td>
<td>1</td>
<td>7/10/64</td>
</tr>
<tr>
<td>5.3.1.5</td>
<td>Relation of Format to the I/O List</td>
<td></td>
<td>11/08/63</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Assembling the I/O List in NICAP</td>
<td>1</td>
<td>7/10/64</td>
</tr>
</tbody>
</table>
5.2 Direct Communication with Back-Up Storage and I/O Tapes

Programs in the monitor area are available for direct communication without any form of conversion. The auxiliary storage units are addressed logically by the user program; the monitor program does a table look-up to get absolute addresses.

Sections 5.2.1 and 5.2.2 describe programs that are part of the permanent monitor. Special uses that are not adapted to these programs may require additional optional monitor programs to be incorporated for that use only.

Programs in the monitor area will run with user interrupt enabled or disabled. If it is enabled, a CALL on the programs is sequenced via interrupt since it is in protected memory; otherwise the transfer is direct.
5.2.1 SYSIØ

SYSIØ is a program for direct communication with the input/output tapes. It includes an input and output buffer so that when control has been returned to the programmer the transfer has been completed as far as he is concerned. The sequence of operations is:

Input: Wait until the input buffer is loaded, then copy information to programmer's area of core. Start a refill of the input buffer and return to the user.

Output: Wait until the output buffer is empty, then copy user's data to buffer. Begin transfer of buffer to output tape and return control to user.

This simple picture is complicated by the fact that records are "blocked," but this does not affect the user.

Use of SYSIØ

The call sequence is

```
CALL SYSIØ
DECO ØP, A, EØF, W
```

ØP determines the operation. Codes are

0  Read a binary card (20 words)
1  Read a BCD card    (10 words)
2  Read a BCD card with a $ in column 1
3  Read a BCD card without a $ in column 1

(The last two options are intended for the system programs!)
512 Punch a binary card (20 words)
513 Punch a BCD card (10 words)
514 Print a line (17 words)

The data input or output starts at location A. On input, sign bits of the quarter words are cleared. On output they are ignored. In the case of print, the first 3/8 of the first word is ignored, the next 1/8 is the carriage control. Thus the format of these 17 words for output is:

\[
\begin{align*}
\text{first word} & \quad \overline{XXX}X\,|\,C\,|\quad | \quad \text{2-17th words} \\
\text{ignored} & \quad | \quad \text{4 characters} \\
\text{carriage control} & \quad | \quad 
\end{align*}
\]

During read operations an end of file may occur. This will cause a branch to the E0F address, if it is nonzero. If the wrong card type is read (e.g., Binary is requested and it is a BCD card, etc.) then a branch to location W occurs if it is nonzero. No read takes place in either of these cases. If the branch address is zero, execution is terminated. Also, SYSIO will not go beyond an E0F mark on the input tape.

**Format of Input and Output Tapes**

The input and output tapes are written in binary with odd parity throughout. Each record is 488 characters or 61 ILLIAC words. (It can be 168 or 328 if desired.) Each record can, but need not, contain three card or line images in the format.

\[
\begin{align*}
\text{1 Record} & \quad \overline{\text{Control Word}} \quad | \quad \overline{20 \text{ words}} \quad | \quad \overline{20 \text{ words}} \quad | \quad \overline{20 \text{ words}} \\
\text{1st image} & \quad | \quad \text{2nd image} \quad | \quad \text{3rd image}
\end{align*}
\]
The control word consists of the eight characters

blank, number of images on this record, blank, T1, blank, T2, blank, T3

where T1, T2 and T3 are the types of the three images

\[ \text{blank} = \text{BCD card} \]

\[ 1 = \text{line for printer} \]

\[ 2 = \text{Binary card} \]

Images are left-justified in the 20 word areas, that is a BCD card occupies the first ten words, the next ten are not used. A line image occupies the first 17 words.
5.2.2 SYSAUX

The system auxiliary storage program SYSAUX enables one block to be transferred to a tape, the drum or the disk. The call sequence is

```assembly
CALL SYSAUX
DECO OP, A, EOP, U
```

where

- `OP` is the operation type
- `A` is the starting core address
- `EOP` is the `EOP` branch address

and

- `U` is the logical unit.

Currently assigned units are:

- `0-N` refer to the drum sectors 0-N. The monitor may relocate this upwards to avoid locked-out areas, so N should not be too large.
- `1024-1033` Tapes 0-9.
- `Tapes 0-5` are system tapes and are not normally available directly. Logical units 6-9 should be used where possible.

Unlike SYSL, there is no buffering in SYSAUX. This has two consequences.

1. On the drum, all transfers are of 256 words. The bottom 8 bits of the address `A` are therefore ignored and a complete block starting from a multiple of 256 is transferred. On tapes, transfer terminates either at an end-of-record gap (reading only) or at the end of the core block of 256 words being used. Therefore care must be exercised in allocating buffer areas for auxiliary transfers.

2. The transfer is overlapped, so that when control is returned to the user, all previous transfers on that unit have been completed successfully without errors, end of files or end of tape signals, and the present transfer has been initiated, but not necessarily completed. If it is necessary to use the information or core area before another
transfer is made from or to the same unit, the control operation WAIT described below should be used. (Efficient programs will avoid the use of this operation!)

If an end of file (not possible on the drum) is encountered, the new transfer is not initiated; instead, a transfer is made to the $EOF$ address if it is nonzero; otherwise execution is terminated. Errors are checked for and the transfer is repeated a number of times until successful or execution is terminated.

Operations for all units

0     Read a block
512   Write a block
256   WAIT until the last block has been transferred

Operations with meaning for tape units

1064  Backspace Record (returns to beginning of last record)
1072  Backspace File (returns to beginning of this file)
1088  Check Record (moves tape to next record)
1096  Check File (moves tape to beginning of next file)
1056  Rewind
1104  Write $EOF$ Mark
5.3 Input Output with Conversion

Input data, normally from cards, and output data, destined for the printer or punch are normally in the IBM 6-bit BCD code shown on page 4.10.3. In order to convert to or from a string of such characters, from or to a suitable internal form in memory, two types of information must be provided:

a) Where the information comes from or goes to in the core memory (the "input-output list").

b) A description of the format of the string of characters to be input or output.

The programming language being used will determine how the input-output list is specified. The format description is essentially independent of the language, as it consists of a string of characters describing the basic fields of the input or output information. This format string is present in the memory at execution time. (It is put there by the appropriate statement in the language used, for example, FORMAT (...) in Fortran and CHR N, ...* in NICAP.) It consists of a sequence of field descriptions which must be paired with consecutive items in the I/O list. The I/O list is a list of addresses of data given in the program. The way in which the I/O list is programmed depends on the user language, and is therefore described in the appropriate section. Section 5.3.1 will describe how the format string is made up.
5.3.1 Format Control

Input and output is by means of 80 column cards or a 132 column printer. The user determines how many of these columns are to be used for the first variable to be input or output, how many for the second variable, and so on, always starting at the left end. The size of these groups of columns, called fields, is determined by a decimal integer in the format string. For example, A17,I19,S5 describes an A field of 17 columns followed by an I field of 19 columns followed by an S field of 5 columns. The meaning of the field is determined by the field description letter appearing before the number. The allowable letters are described in the following sections.

# In the case of output to the printer, the printer removes the first (left most) character transmitted to it and uses it for carriage control. Hence a maximum of 133 characters may be transmitted to the printer per line.
5.3.1.1 Field Descriptions

In the descriptions below, w and n are unsigned decimal integers, and d is an unsigned decimal digit. Inputs are described in terms of reading cards, but also apply to reading card images on magnetic tape. Outputs are described in terms of printing, but also apply to punching cards or writing magnetic tape.

F\(\cdot\)d or I\(\cdot\)d The I or F fields, which are identical, transmit a decimal number with the decimal point shown, for example 15, 101.66, -.0034, and 34352. The number is transmitted to or from a full word location, hence the corresponding entry in the Input-Output List must refer to a full word. The w indicates the total width of the field in columns. The d indicates the number of places to the right of the decimal point.

On input, d is ignored if there is a decimal point in the number, blanks are ignored, and an absent sign is assumed to be plus. A decimal point may be omitted, in which case it is assumed to be d places from the right end of the number. If w is larger than needed, the number may be punched anywhere in the field.

On output, the number is printed with d places after the decimal point, right justified in the w column field. Leading zeros are suppressed up to the last digit before the decimal point. Plus signs are suppressed and minus signs are printed. (To print plus signs also, write F+w.d.) If the number to be printed is floating zero, then 0 with the appropriate decimal point and trailing

Floating zero is represented in the core memory as 0 \cdot 4^-64.

Date: 7/10/64
Section: 5.3.1.1
Page: 1 of 6
Change: 1
zeros will be printed. If the output number overflows
the field on the left end, then the number will be printed
in E form (even though F was specified) with the least
significant digits of the fraction part truncated if
necessary to fit into the required number of columns.

Fw or Iw
This indicates that a decimal integer is to be input
or output. The w indicates the total width of the field
in columns. On input, blanks are ignored. On output, the
number is printed right justified in the w column field
with leading zeros suppressed and with plus signs sup-
pressed, and without a decimal point. (To print the plus
signs, write F+w or I+w.) If the integer to be printed
is too large, it is printed in E form as described below.

Ew.d
The Ew.d field describes a number which resembles the
form known as "scientific notation," for example,
\(0.23443 \times 10^2\). Since card readers and printers do not
handle superscripts, the exponent is indicated by pre-
ceding it with an E. The general form of a number in an
E field is

\[ \pm \cdot xxx \ldots \times e^{ee} \]

where the x's represent decimal digits, the E implies
"exponent follows," and the ee represents a two digit
exponent of 10. The w indicates the total width of the
field in columns. The d indicates the number of places
after the decimal point. (There are none before the point
unless the field description is modified by a P control
character, described below.) Note that up to six columns
of an E field are used for the characters

\[ \pm \cdot e^{ee} \]

so that w must be greater than d by 6 or more. The number
in an E field is transmitted to or from a full-word loca-
tion, so the corresponding entry in the I/O list must refer
to a full word. The range of exponents is -38 to +38 unless modified by the P control character (see below).

On input, the number may have the general form indicated above, or an abbreviated form as indicated in the following section. On input, d is ignored.

On output, the number will have the general form shown above, except that the sign of the number is printed only if minus. (To print plus signs also, write E+w.d.) The sign of the exponent is always printed and the exponent is always printed, even if zero. If the field size, w, is larger than required for the information, the number will be printed right justified in the field, with blanks supplied on the left. Floating zero will be printed as

.0------0E-99

For example, using the field description E10.4, the decimal number +10.39 would be printed as

b.1039E+02

where b stands for blank. Using E9.3, the same number would be printed as

b.104E+02

Using E9.3, the number -10.39 would be printed as

-.104E+02

Variations allowed on input

On input, E, F, and I fields will accept any of the following forms

109    unsigned integer
+11    signed integer
7.1    unsigned number with decimal point
-8.132 signed number with decimal point
or any of the above followed by one of the following forms, which indicate the decimal exponent:

- E-07  general form
- E12  plus sign absent
- E+3  leading zero in exponent absent
- +3  E and leading zero absent
- -21  E absent

Note that E is necessary only if the sign of the exponent is not punched. Unless modified by the P control character (see below), E, F, and I fields are identical for input. On input, blanks are ignored in the field. If the entire field is blank, the value will be set equal to floating point zero. Any number of digits may be used in the field, but only 13 decimal digits of accuracy are retained.

Sw  Space. On output, Sw causes w spaces to be printed. On input, Sw causes w columns to be ignored.

X  is identical to Sl.

An or Cn  These field descriptions are used to transmit Hollerith characters in the 6-bit BCD code given on page 4.10.3. The field description An or Cn causes n Hollerith characters, packed 8 per word in the least significant 12 bits of each quarter word, to be input or output. For example, if the field description A10 were used to read a card punched with

```
ABDEFGHIJ
```

then the two computer words involved would contain

```
1 6 6 1 6 6 1 6 6 1 6 6

A B C D E F G H

```

The first bit of each quarter word is unchanged. If n is

```
#Before the program was loaded, the memory was cleared to zero in every bit position. Unless the program has changed it, these bits are still zero.
```
not a multiple of 8, the remaining space in the last word is not changed.

Dn

This field description transmits quarter words as decimal integers. When a D field is used, the corresponding item in the Input-Output List should refer to a quarter word.

On input, the quarter word specified in the I/O list is loaded with the number truncated to an integer modulo 8192. Numbers greater than 4095 may be thought of either as positive or twos complement negative number. Thus 4097 is -4096+1, 4098 is -4096+2, ... 8191 is -4096 + 4095 = -1.

On input, the number may have any of the forms allowed for E fields.

On output, a decimal integer in the range -4096 to +4095 is formed, so at least five columns are needed. If n is greater than five, blanks are inserted on the left.

Qn

This field description transmits quarter words as unsigned octal integers, in the range 00000 to 17777. When a Q field is used, the corresponding item in the Input-Output List should refer to a quarter word.

On input, blanks are ignored, other characters are assembled as octal, but no check is made that these characters are octal digits.

On output, five octal characters are printed, right justified in the n column field. No zero suppression occurs. If n is less than five, only the rightmost digits are printed. If n is greater than five, blanks are inserted on the left of the five octal characters.

In

An I field is similar to a D field in that it transmits a 13-bit integer as a decimal integer.
On output, the address given in the I/O list is printed, not the contents of the addressed word. A decimal integer in the range \(-4095\) to \(4096\) is printed as for the D field so at least five columns are needed.

On input, the number read is copied into the read program image of the Input/Output List. The Programmer's I/O list is not changed; but if this address is used for the next item, the changed value is used.

An M field is the same as an L field, except that the 13-bit address is input or output as an octal integer in the range \(00000\) to \(17777\). L and M fields are provided for the benefit of dump programs. It is doubtful if they will have much value in general programming.

Double Precision

Floating-point conversion (E, F and I fields) is performed almost correct to double precision (rounding will depend on the exponent). Double precision words may be input and output by doubling the field description letter. Thus EE30.24 will read or print a 24-digit number in a 30-place field. On input, the most significant part will go into the cell named in the I/O list, the least significant part into the next higher addressed cell. It counts as only one item in the list count but increases the address by 2. On output, the contents of the two cells are added, and then converted double precision.
5.3.1.2 Multiple Field Descriptions

Any field description letter may be preceded by a decimal integer which indicates the number of times that field is to be repeated. Thus 4I7,9X,3Q6 indicates four I fields of seven columns each, then 9 spaces, then three Q fields of six columns each:

```
I   I   I   I   blank   Q   Q   Q
7   7   7   7   9   6   6   6
```
5.3.1.3 Hollerith Fields

One field description letter must always be preceded by a number, the Hollerith field. The general form of this field description is \( nHx_1x_2x_3\ldots x_n \) where the \( x_i \) represent characters.

On output the \( n \) characters in the format string which immediately follow the \( H \) are to be printed exactly as shown. Note that in this case all the information to be printed comes from the format string itself, not partly from the format string and partly from the Input-Output List as is the case for other format specifications.

For example, the format string \( 10Hb\text{AVERAGE}b=,F7.1 \) will cause the 10 characters, including blanks, following the \( H \) to be printed, followed by the number corresponding to the \( F7.1 \) description (as indicated by the I/O list). If that number happens to be +101.3, the printed output will be

\[ b\text{AVERAGE}b=bb101.3 \]

On input, \( n \) characters are read from the input medium and stored (in 6-bit BCD form) as the next \( n \) characters in the format string itself as it is stored in the memory.
5.3.1.4 Control Characters

Except for H fields, each field description must be followed by one of the characters

, / ) *

Their meanings are

, is solely a separator.

/ means output this line and go on to the next or read next card.

* means "end of format statement." It must be given explicitly in NICAP. In FORTRAN it is supplied automatically.

Parentheses "(" and ")" can be placed around a valid sequence of field descriptions, and a decimal integer can appear immediately before the ")". This means that the descriptions enclosed in the parentheses are to be repeated the stated number of times. If no number appears before the "(," it is assumed to be 1. Thus 3(F5,2HX=2E15.6/) will print a five-digit fixed-point number, X=, and two 15-column floating-point numbers followed by a "carriage return" and repeat this three times.

Parentheses may be nested to a depth of three levels.

P Power or scale factor. The general form is nP where n is a signed or unsigned decimal integer. The integer is a scale factor that has the following effect on the next E, I or F field.

a) E Fields No effect on input. On output, it is the number of places before the point. It does not affect the size of the number, i.e., the exponent printed is adjusted to compensate for the shifted decimal point.

b) I and F Fields It scales the field by \(10^{-P}\) on input, and by \(10^P\) on output.
Thus for F and I fields the relation

$$\text{External number} = \text{Internal number} \times 10^P$$

holds for input and output.

Example:

Using the format \(-1P,E+13.7,F7.4\), ... the internal numbers 10.13 and 17.25 would print as

\[+.0101300E+03b17.250\ldots\]

If a P is not given, the scale factor is taken as zero. The scale factor is cleared to 0 after a single or multiple field description has been processed, e.g., \(2P,3E17.5\) would scale three fields by 100.

**Indirect Address Character \(\cdot N\)**

At any point where a decimally represented number may appear in a format statement, an N may appear in its place. The control program reads the next full word specified by the I/O list from memory, converts it to an integer by truncation and uses that number for the format control. Thus ..., NE15.6, ... will print a number of floating-point words specified by a cell named in the I/O list.

**Blank Characters**

Blanks may appear anywhere in the format specification. They have no effect.
5.3.1.5 Relation of the Format to the I/O List

The format is scanned first and each time that it requires a word location (E, F, I, and A fields), a quarter word location (D and Q fields), or an address (L and M fields), the I/O list is examined. If there are no items left on the list, the input or output is terminated by a card feed or a line eject. If there is an item, it is used. When the end of the format (*) is reached, a card feed or line eject is given and the I/O list is checked. If it is empty, again the input output is terminated. If it is not empty however, the format definition is continued by repeating from the last occurring outermost left parenthesis, or from the beginning if there are no parentheses. The combination of format and I/O list must not cause more than 80 columns to be transmitted to or from a card, or more than 133 characters to the printer. The printer only has 132 print positions; the first character transmitted is removed during printing for carriage control. Assignment of carriage control characters will be given in Chapter 2. Standard ones will always be

blank--single line feed
l--page eject Before printing.

The blank can be provided most simply by making sure that the first field is sufficiently long to provide a blank.
5.3.2 **Assembling the I/O List and Format Statement in NICAP**

The format statement can be most easily assembled by use of the CHR pseudo operation (see section 4.5.2, page 1). For example, to store a format statement in location G, write

\[
G \text{ CHR } 18,4F19.1,3HX1=E21.8 *
\]

This would assemble as three consecutive words in G, G+1, and G+2.

The I/O list is stored in a set of consecutive memory locations. First let us consider full word items. Suppose that the data in locations A, A+1,...A+N-1, B, B+1,...B+M-1; and C, C+1,...C+P-1 is to be transmitted. Then the I/O List will consist of three consecutive words in memory, thus:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>112</th>
<th>13</th>
<th>25</th>
<th>26</th>
<th>38</th>
<th>39</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>1</td>
<td>00</td>
<td>A</td>
<td>N</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List+1</td>
<td>1</td>
<td>00</td>
<td>B</td>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>List+2</td>
<td>00</td>
<td>00</td>
<td>C</td>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The 1 in bit position 0 of the first two words indicates that another control word follows, i.e., this is not the end of the Input-Output List. The 00 in bit positions 0 and 1 of the third word indicates that this is the last control word. Quarter word 1 (bits 13-25) of each control word contains an address. Quarter word 2 contains a count of the number of items to be transmitted beginning at the address in quarter word 1. This count must be less than 4096. After the specified number of items have been transmitted, the input-output program examines the next control word to see what item is to be transmitted next, terminating at the end of the control word which begins with 00. Quarter word 3 of the first control word contains the address of the format string (the address of the first word in the string if it occupies several words). Quarter word 3 of all the other control words is ignored.
The PRINT, READ, and PUNCH programs are entered by storing the address of the first word of the I/O List in M1, and then CALLing the appropriate program. For example, to print the items indicated by the I/O List above, write

```
CAM    1,LIST
CALL   PRINT
```

The CALL pseudo order will be assembled as JSB3,,PRINT.

For example, to print A, A+1, A+2, A+3, and B with the format statement used earlier, the program below could be used

```
CAM    1,H1                  In Program
CALL   PRINT
...
G     CHR   18,4F19.1,3HX1=E21.8*  Constants
H1    DECQ  4096,A,4,G        elsewhere
       DECQ  0,B,1,0            in memory
```

A little care is needed because DECQ is a quarter word pseudo operation, i.e., it does not FIL before loading. Therefore it should either be preceded by a full word pseudo operation, as in this example, or should be preceded by a FIL.

When a quarter word address is required, as in D and Q, the last two bits, indicating quarter 0 to 3, are stored in the least significant two bits of the first quarter of the control word (bits 11 and 12). Thus to PRINT locations A,3; A+1,0; A+1,1; and B,2; B,3; and B+1,0 in octal, the program might be

```
CAM    1,C
CALL   PRINT
...
FIL
C     DECQ  4096+3,A,3,F
       DECQ  2,B,3,0
F     CHR   4,6Q7*
```
Note that the item count in quarter word 2 is the number of quarter words. In general, this number is the number of items printed where each E, F, I, A, C, D, L, M, and Q field counts as one item. H, S, and X do not. If a full word item is input or output when the current I/O list address (i.e., that one to be used next) indicates quarter word 1, 2, or 3 rather than zero, then the quarter-word address is reset to 0 and the full word address is incremented by 1 before the input or output. Thus

```
CAM    1,D
CALL   READ
FIL
D      DECQ 4096+2,A,4,Fl
DECQ   1,B,1,0
Fl     CHR   11,3Q6,2E20.0*
```

will read three octal numbers into A,2; A,3; and A+1,0, and two full-word floating-point numbers into A+2 and B+1.

If the bits 0 and 1 of the last control word are the 01 combination, then this indicates a "partial CALL." The next CALL either of the READ, PRINT or PUNCH program will be interpreted as a continuation of the previous CALL. That is, the same program will be used as was used on the previous occasion (it does not matter whether PRINT, READ or PUNCH is called), and the old format will be continued from the point it has previously reached. In other words, it is equivalent to placing the new I/O list on the end of the last one.

**Example:**

To print the first 976 integers, ten per line, the following program could be used:
CAD  1.
STR  A
CSM  4,976
CAM  1,PR+1

B CALL PRINT
CAD  1.
ASC  A
CJU  4,B
CAM  1,PR+2
CALL PRINT

----- ----------

PR CHR  8,10F10*
DECQ  2048,A,1,PR
DECQ  0,0,0,0

Date: 7/10/64
Section: 5.3.2
Page: 4 of 4
Change: 1
7.1 FORTRAN II Version I

7.1.1 Characters Used  6/3/65
7.1.2 Source Program Card Layout  6/3/65
7.1.3 Constants  6/3/65
7.1.4 Variables  6/3/65
7.1.5 Subscripted Variables  6/3/65
7.1.6 Expressions  6/3/65
7.1.7 Statements  6/3/65
7.1.8 Subprograms: Functions and Subroutines  6/3/65
   7.1.8.1 How to Name a Function  6/3/65
   7.1.8.2 How to Define a Function  6/3/65
      7.1.8.2.1 Arithmetic Statement Functions  6/3/65
      7.1.8.2.2 FUNCTION a(a1, ..., an)  6/3/65
      7.1.8.2.3 User Defined Library Functions  6/3/65
   7.1.8.3 How to Name a Subroutine  6/3/65
   7.1.8.4 How to Define a Subroutine  6/3/65
   7.1.8.5 How to Use a Subprogram  6/3/65
7.1.9 Printed Output From a Compilation  6/3/65
7. COMPILERS

7.1 FORTRAN II, Version I

Version I of the FORTRAN II compiler for ILLIAC II (hereinafter called FORTRAN) is designed to be fast at compiling at some expense in object code efficiency. Generally speaking, the code generated is the best which can be generated in one pass, making no special tests for restricted cases. Thus index registers are not used in DO loops because at least 15 bits are generally needed, and multiplication for array indexing is not moved outside of DO loops. However, if subscripts are specified as numbers, the computation is done at compile time. A detailed description of the compiler forms Section 3.5 of the ILLIAC II Systems Manual.

Logically, the ILLIAC II FORTRAN compiler is a "one pass to assembly language" compiler. The output of the FORTRAN pass is fed directly into Pass II of the NICAP assembler. NICAP then produces a relocatable binary object program (plus a listing, if desired--see 7.1.9).

FORTRAN II for ILLIAC II is essentially compatible with FORTRAN II for the IBM 7094 as implemented under the PÔRTHOS operating system at the University of Illinois. There are certain differences, however. In particular, FORTRAN II for ILLIAC II will permit mixed arithmetic expressions (i.e., both floating and fixed point quantities in the same expression); the statements READ DRUM and WRITE DRUM are ignored by ILLIAC FORTRAN; and PRINT n, list causes a line image for off-line printing to be written on tape. Further, at present, none of double precision arithmetic, complex arithmetic, or Boolean arithmetic (D, I, and B respectively in column 1) are implemented for ILLIAC FORTRAN II. Cards in these categories will be ignored in compilation, but will appear on the listing, and will generate nonfatal errors.
This brings us to another item in the philosophy of ILLIAC II as applied to FORTRAN: every effort is made to compile errors. Therefore a distinction is made between fatal errors and nonfatal errors. Fatal errors cause an unsuccessful compilation. However, the compiler will continue to analyze statements after finding a fatal error in the hope of finding more errors. Nonfatal errors need not cause an unsuccessful compilation; the compiler can make some sense out of any statement containing nonfatal errors and no fatal errors, and it will compile the object code that it decides the programmer wants. Nonfatal errors are also generated, as has been mentioned, by such things as mixed expressions, even though in this case there is no doubt about what the programmer wants.

Control cards must precede all programs run under the ILLIAC II Operating System including FORTRAN programs. The user is referred to Chapter 2 of the present manual for details of the operating system. We mention here some of the necessities:

1. Each complete program must be preceded by a single ID card, whose format is identical to the format of the ID cards used by the IBM 7094 operating under FORTHOS.

2. If execution is desired after compilation, a $ GO card must be included between the ID card and the complete program.

3. Each program or subprogram written in FORTRAN must be preceded by a $ FORTRAN card.

4. If data is to be read from cards, it must be preceded by a $ DATA card.

The remainder of Section 7.1 is devoted to a more detailed explanation of FORTRAN. It is intended to be used as a reference, rather than as a learner's manual. As a reference, it is reasonably complete, perhaps more so than would be desired by people who are already familiar with FORTRAN in general and want only to get at the peculiarities of this version of FORTRAN. We apologize to such people, and hope they will be able to find what they want in spite of having to wade through much material.
Sections 7.1.1 through 7.1.6 specify in part the terms which can appear in FORTRAN statements. More details are given in Section 7.1.7, where the statements are explained in alphabetic order, and in 7.1.8, where subprograms are explained. Section 7.1.9 explains the printed output from a FORTRAN compilation.
7.1.1 Characters Used

The (decimal) digits are 0 through 9; the letters are A through Z. The alphanumerical characters are the letters and digits together. The special characters used are = + - * / ( ) , .. Two other special characters, $ and ' may be used in comment cards and in H field specifications in Format statements.
7.1.2 Source Program Card Layout

The standard IBM FORTRAN card layout is used. Thus, for example, there may be as many as nine continuation cards, columns 1 through 5 are allowed for the statement number (which must however be between 1 and 32,767 inclusive), and the statement starts in column 7. Columns 73 through 80 are ignored. The letter C in column 1 identifies a comment card; it is ignored in compiling, but appears on the program listing. Comment cards may appear anywhere in a program. An F in column 1 has a special meaning (see 7.1.8). All other cards should be blank in column 1; any character other than F, C or blank (in particular, a D, B or I) in column 1 causes the card to be ignored.
7.1.3 Constants

1. Fixed Point Constants: have 1 to 13 decimal digits, absolute value less than \(2^{14}\).

2. Floating Point Constants: are real numbers, absolute value between \(10^{-38}\) and \(10^{+38}\), which must be written with a decimal point (e.g., 5.0, -.01). They may be followed by E and a decimal integer \(n\), denoting multiplication by \(10^n\) (e.g., \(3.1E-6 = 3.1 \times 10^{-6}\)).

Note: 1. Fixed and Floating point constants are represented in an identical manner inside ILLIAC II. Thus arithmetic statements and expressions can contain both kinds of constants and variables, contrary to the usual rules of FORTRAN. That is, \(A = B + 1\) and \(A = B + 1\) have the same effect. Every constant occupies one word of memory (double precision is not available) of which the left-hand 45 bits represent the mantissa (with sign) in two's complement and the right-hand 7 bits represent a base 4 exponent, also in two's complement notation.

2. Constants which are out of the allowed range will compile with the largest allowed value, and will cause a nonfatal error to appear on the listing.
7.1.4 Variables

1. **Fixed Point Variables**: have 1 to 6 alphanumeric characters, beginning with I, J, K, L, M or N. They have fixed point values and have the same range as fixed point constants.

2. **Floating Point Variables**: have 1 to 6 alphanumeric characters beginning with any letter except I, J, K, L, M or N. They have floating point values with the same range as floating point constants.
7.1.5 Subscripted Variables

Fixed and floating point variables may both have as many as three subscripts attached e.g., AAAAA(3), Alllll(I, K, 4) are floating point subscripted variables; IIIII(99), I22222(L M, 49) are fixed point subscripted variables. Subscripts should be either fixed point constants or fixed point variables; if a floating point quantity occurs as a subscript, it will be flagged as a nonfatal error.

Subscripted variables must appear in a DIMENSION statement (see 7.1.7) before they appear anywhere else in a program, including in a COMMON statement, except that they may appear as call parameters in the first line of a subprogram (see 7.1.8) before appearing in a DIMENSION statement in the subprogram.

Subscripts may be any valid arithmetic expression, except for input and output, where only the following kinds of expressions are allowed: (C and C' are unsigned constants and V is a nonsubscripted variable).

\[ C, \dot{V}, V + C, V - C, C \times V, C \times V + C', C \times V - C' \]

In particular, expressions such as \( C + V \), \( C \times C' \) are not allowed as subscripts for input and output.

In arithmetic expressions subscripted variables can themselves be subscripted; this nesting of subscripts can be used to any level. For example \( H40K (K, NOT (NOT (K))) \) is legal.

If a subscript is greater than 4095, an overflow will occur, and execution will be terminated by the system unless a system program that inhibits overflow traps has been called.
7.1.6 Expressions

1. Arithmetic Expressions are strings of operands, operators, and brackets such as $A + B/C$, $I1 + (2.*J1 - K)$, $A * * B * * C$. The operands in these expressions are "A", "B", "I1", "2.", "J1", and "K". The operators in these expressions are "+" (addition), "/" (division), "*" (multiplication), "-" (subtraction), and "**" (exponentiation). These are the only operators which are allowed.

The operands in a given expression should be either fixed or floating; if both types are included, the expression is called a mixed expression and will be flagged as a nonfatal error, except that floating ** fixed will not be flagged. Any mixed expression will be treated as a floating point expression.

The usual operator priority rules are observed, i.e., exponentiation is performed first, multiplication and division next, and addition and subtraction last. If brackets are omitted, they will be inserted from the left. Thus, for example, $A ** B ** C$ is evaluated as $(A ** B) ** C$, and $A * B/C/D ** C$ is evaluated as $((A * B)/C) / (D ** C)$.

Note: All floating point operations are rounded rather than truncated so that numerical answers will generally be more accurate than those obtained with the same program when it is compiled by a compiler which uses a truncation process. In general, expressions are not rearranged for more efficient computation.

2. Boolean Expressions are not yet implemented.

3. Hollerith Expressions such as JHYES are not yet implemented except, of course, in FORMAT statements for output.
7.1.7 Statements

First, we define the word "List," which is used in explaining input/output statements. A list is a sequence $x_1, \ldots, x_n$ where each $x_i$ is one of the following:

1. A fixed or floating point variable, which may be subscripted, e.g., A; INTGR; A(3); X(4,9); ALP(I,2).

2. A sequence of subscripted fixed or floating point variables, with variable subscripts, possibly followed by an expression giving the ranges of the variable subscripts, all enclosed in brackets: e.g., (A(I), I = 1, 3); (C(K,3), X(I,K,4), K = 3,9); ((A(I,J), I = 1,10), J = 1,5). In the second example, I must be a variable with a previously assigned value.

3. An array name.

Variables are input or output in the order in which they appear in the list, and within that order, they are ordered as illustrated by the following: e.g., "A, (B(K,I), K = 1, 3), ((D(L,M), M = 1,2), L = 2, 3), H" is a list.

Suppose A is an array with dimension (2,3). Then the variables occurring in the list will be input or output in the following order: A(1,1), A(2,1), A(1,2), A(2,2), A(1,3), A(2,3), B(1,1), B(2,1) B(3,1), D(2,1), D(2,2), D(3,1), D(3,2), H.

A list of FORTRAN statements, together with some explanation of the meaning and use of these statements, follows. The statements are listed in alphabetical order.

$a = e$: This is an arithmetic statement. $a$ must be a variable (either fixed or floating), and $e$ must be an arithmetic expression (which may involve the variable $a$). Boolean statements are not yet implemented. $a = e$ will result in the contents of the location whose name is $a$ being set equal to the value of the expression $e$. If $a$ is a floating point variable and $e$ is a fixed point expression, then the value of $e$ will be converted to floating point before being stored in $a$, and conversely, if $a$ is a fixed point variable and $e$ is a floating point expression, then the value of $e$ will be converted to fixed point before being stored in $a$.
Example: \( X = X + 1 \). This will result in the contents of \( X \) being replaced by what is now in \( X \), plus 1.

ASSIGN \( n \) \( \rightarrow \) \( i \): \( n \) must be a statement number, and \( i \) must be a nonsubscripted fixed point variable which is not an array name. \( i \) must appear in an assigned \( G \) \( \rightarrow \) \( T \) statement, and \( n \) should appear in an assigned \( G \) \( \rightarrow \) \( T \) statement. Note that the compiler does not check to see that this last condition is satisfied.

Example: ASSIGN 47 \( \rightarrow \) INT\( T \) will cause INT\( T \) in \( G \) \( \rightarrow \) INT\( T \), (16, 47, 36, 9) to have the value 47.

BACKSPACE \( j \): causes symbolic tape unit \( j \) to backspace one logical record (a logical record is defined to be the physical records written by a previous WRITE TAPE statement). At the moment only scratch tapes 6 and 7 are available, but eventually more possibilities (i.e., more tape units) will be available for \( j \). Note that no check is made at compile time to see that \( j = 6 \) or 7.

CALL \( a(a_1, \ldots, a_n) \): see 7.1.8, 7.1.8.5.

COMMON \( x_1, \ldots, x_n \): Each \( x_i \) is a fixed or floating point variable or array name \((i = 1, \ldots, n)\). Array names must appear in a DIMENSION statement before the COMMON statement in the same program as the COMMON statement.

COMMON is used in a calling program and in a subprogram to enable both programs to gain access to certain quantities in one area of memory called COMMON.

As each program is compiled, a counter is used to keep track of how much of COMMON has so far been used. This counter is set to zero at the beginning of each compilation, and is increased by one by each variable or position of an array put into COMMON in the course of the compilation. Thus each variable or position of an array specified by a COMMON statement is associated with a unique number by this counter. When the programs are consolidated at run time into one program, every COMMON variable or position of an array associated with the same number is assigned to the same physical location in memory. This is the way in which variables and arrays in COMMON are used by several subprograms.
Example: If the first COMMON statements in each of two subprograms are

\begin{verbatim}
COMMON X1, X2
COMMON X3
\end{verbatim}

and

\begin{verbatim}
COMMON R1, R2, R3
\end{verbatim}

respectively, this would result in \(X_i\) and \(R_i\) representing the same location (locations, if \(X_i\) and \(R_i\) are array names of the same size), \(i = 1, 2, 3, \ldots\).

Any variables in COMMON which also appear in an EQUIVALENCE statement are located at the beginning of the COMMON area i.e., the EQUIVALENCE statement alters the number associated with a variable by the counter mentioned above.

Example: The statements

\begin{verbatim}
COMMON A, B, C, D
EQUIVALENCE (C, G), (E, B)
\end{verbatim}

will cause A, B, C and D to be stored in COMMON in the order C, B, A, D rather than in the order A, B, C, D. G and E will be stored in the same locations as C and B respectively, as specified by EQUIVALENCE.

If the READ TAPE or WRITE TAPE statements are used, then the first 256 words of COMMON must be allocated for buffer use (see the library program I/O LIST). This can be accomplished by

\begin{verbatim}
DIMENSION XXX(256)
COMMON XXX
\end{verbatim}

at the start of each program segment where XXX is a variable not used elsewhere in the program. The COMMON area of memory is located at the beginning of user's core.

CONTINUE: is a dummy statement used to end a DO loop if the DO loop would otherwise end with a transfer or a nonexecutable statement.
DIMENSION $a_1, \ldots, a_n$: (see also 7.1.5). Each $a_i$ is a subscripted variable with one, two, or three numerical subscripts. The values of the subscripts given here are fixed point constants equal to the maximum value of the subscripts used in the rest of the program, e.g., $a_1 = \text{MATRIX}(10, 20, 15)$ defines a three-dimensional matrix of dimension $10 \times 20 \times 15$. The total amount of storage specified in one DIMENSION statement must be less than 4096 words.

The unsubscripted variable appearing in each $a_i$ is referred to as an array name, e.g., MATRIX is the array name in the example above. If an array name appears in a program, it is understood to refer to the first location in the array, e.g., MATRIX refers to MATRIX(1,1,1). Note that an array name is not considered to be an unsubscripted variable.

A DIMENSION statement sets aside one word in memory for each of the elements in the array.

Arrays are stored in forward order in memory. For example, the $2 \times 2 \times 2$ array $A$ is stored in successively higher numbered locations in the order $A(1,1,1), A(2,1,1), A(1,2,1), A(2,2,1), A(1,1,2), A(2,1,2), A(1,2,2), A(2,2,2)$. The unique DIMENSION statement containing a given array name must appear before that array name is used elsewhere, except as mentioned in 7.1.5.

DO $n = i_1, j_2, j_3$: $n$ must be a statement number referring to a statement following the DO statement; $i$ must be a nonsubscripted fixed point variable; $j_1$, $j_2$ and $j_3$ must be either fixed point constants or nonsubscripted fixed point variables. A DO statement results in the repeated execution of the DO loop (the statements following the DO statement up to and including statement n), starting with the index, i, equal to $j_1$ increasing i by $j_3$ each time the DO loop is executed, and stopping the repetitions immediately after the DO loop has been executed for the least value of i such that $i + j_3 > j_2$. Note, however, that the DO loop is always executed at least once, even if $j_1 > j_2$. 

Date: 6/3/65
Section: 7.1.7
Page: 4 of 11
Change: ILLIAC II MANUAL
Examples: (a) \[ \text{DIMENSIÓN A(20)} \]
\[
\begin{align*}
I & = 2 \\
J & = 20 \\
\text{DO 10 K = 1, J, I} \\
10 & A(K) = K ** 2
\end{align*}
\]
will result in \(A(K)\) being set equal to \(K^2\), \(K = 1, 3, 5, \ldots, 17, 19\).

(b) \[ \text{DO 13 KAPPA = 3, 2} \]
\[
13 \quad A \quad = \quad \text{KAPPA}
\]
will result in the floating point constant 3 being stored in \(A\).

The \text{DO} statement may also be specified in the form \(\text{DO n i = j_1, j_2}\) in which case it will be assumed that \(j_3 = 1\).

\(i, j_1, j_2, j_3\) should not be altered by any statements in the \text{DO} loop. However, altering \(i, j_2\) or \(j_3\) will be regarded as a nonfatal error. Altering \(j_1\) has no effect. Care must be taken to see that \(j_3\) is not set to zero, and that \(j_2\) is not increased by more than \(j_3\) each time through the loop. It must be true that \(j_1 \geq 0, j_2 \geq 0,\) and \(j_3 > 0\).

Note that the last statement of a \text{DO} loop cannot be a nonexecutable statement (e.g., \text{DIMENSIÓN}, \text{COMMON}), nor can it be any transfer (e.g., \(G \text{ TO}\)) or any \text{DO} statement. If the last statement in the \text{DO} loop would be one of these, the statement

\[
n \quad \text{CONTINUE}
\]
where \(n\) is the statement number appearing in the \text{DO} statement, should be written following what would otherwise be the last statement. Aside from these restrictions, any \text{FORTRAN} statement may appear in a \text{DO} loop.

Example:

\[
\begin{align*}
\text{DIMENSIÓN} & \quad A(10), B(5) \\
\text{DO 5 NICK} & = 1, 5 \\
A(2*NICK) & = \text{NICK} \\
B(\text{NICK}) & = \text{NICK} ** 3 \\
\text{PRINT} & \quad 2, A(2* \text{NICK}), B(\text{NICK}) \\
2 & \quad \text{FORMAT (1HO, 2(I10))} \\
5 & \quad \text{CONTINUE} \\
\end{align*}
\]
\text{END}
This program will cause the following numbers to print out in the format shown (b represents a blank):

```
bbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb8
bbbbbbbbbbbbbbbbbb27
bbbbbbbbbbbbbbbbbb64
bbbbbbbbbbbbbbbbbb125
```

A sequence of DO statements is said to be nested if the DO loop of each DO statement in the sequence contains the next DO statement in the sequence and its DO loop. The length of such a sequence is called the depth of the nest, and is unbounded. Overlapping DO loops are not permitted.

**Examples:**

(a)  
```
A = 1
DO 1 I = 1,5
DO 1 J = 1,5
A = I * J * A
1   B = I + J + B
```

is a nest of DO loops of depth 2 which will result in A being set equal to 6192 \times 10^{17} (computed by ILLIAC) and B being set equal to 150.

(b)  
```
DO 1 I = 1,10
DO/ 2 J = 1,10
1   I = I
2   J = J
```

is illegal, since the two DO loops overlap.

Control can be transferred by means of any GO TO or IF statement from inside a DO loop to outside the loop. The value of the index of the loop is available outside the loop. A nonfatal error will occur if the program transfers back into the range of the DO loop. When such a transfer is made, the value of the index will be the same as it was when the program transferred out of the DO loop, unless the program changes it, in which case it will have whatever value the program gives it.
However, execution of a subprogram inside a DØ loop when the subprogram contains a DØ loop with the same index as the original DØ will not cause the index of the original DØ loop to be changed. The general rule is that all variables in one program with same name (including indices of DØ loops) are identified (i.e., stored in the same location) but that variables with the same name occurring in different programs are not so identified.

END: ends compilation of any program or subprogram. If no END statement is present, the FØRTRAN compiler will generate one. It is compiled as a CALL SYSTEM. In a subprogram, the END also acts exactly as a RETURN statement. If the user wants to terminate execution at any point in his program, he may do so by writing

        GØ TØ n

and prefixing the statement number n on the END statement of his program.

END FILE j: causes an end of file mark to be written on symbolic tape unit j. At the moment, j must equal either 6 or 7.

EQUIVALENCE (x₁, ..., xₙ), (y₁, ..., yₙ), ..., (z₁, ..., zₙ): causes the variables x₁, ..., xₙ to be stored in the same location, y₁, ..., yₙ to be stored in the same location, ..., z₁, ..., zₙ to be stored in the same location. Each x₁, y₁ or z₁ can be fixed or floating point, and may optionally include one subscript, which must be an unsigned fixed point constant, whose meaning is best explained by an example: Suppose x₁ = A(3). Then if A is an array name, x₁ refers to second location following A(1), A(1,1), or A(1,1,1) as the case may be. If A is not an array name, x₁ refers to the second location after A. An array name without a subscript refers to the first element of the array as usual.

If a position of an array is equivalenced to a location X (i.e., either to a variable or to a position of another array), then the whole array will automatically be equivalenced to the locations on either side of X.

Example: if the statements

        DIMENSIØN A(3,2), X(2,4)
        EQUIVALENCE (A(5), X(4))
appear in a program, then the arrays A and X will be stored overlapping each other as given by the figure:

\[
\begin{array}{cccc}
A(1,1) & A(2,1) & A(3,1) & A(1,2) \\
X(1,1) & X(2,1) & X(1,2) & X(2,2) \\
A(2,2) & A(3,2) & X(1,3) & X(2,3) \\
X(1,4) & X(2,4) & & \\
\end{array}
\]

The locations in the box are the locations specified by the equivalence statement.

EQUIVALENCE has roughly the same effect within one program as COMMON has between two or more programs.

FÔRMAT: See 5.3. The FÔRTRAN format specifications for ILLIAC II are not the usual FÔRTRAN format specifications; they correspond to the format specifications for NICAP.

FREQUENCY: ignored.

FUNCTION a(a_1, ..., a_n): see 7.1.8.2.2

GØ TØ n: results in a transfer to the statement numbered n.

GØ TØ (n_1, ..., n_k), i: This is a computed GØ TØ. It results in a transfer to the statement numbered n_i. Thus i must be a nonsubscripted fixed point variable and its value must lie between 1 and k.

Example: If I has the value 2, GØ TØ (3, 39, 14), I results in a transfer to statement 39.

GØ TØ i, (n_1, ..., n_k): This is an assigned GØ TØ. It is generally not as useful as a computed GØ TØ. It results in a transfer to the statement numbered n_r when i = n_r. i must be a nonsubscripted fixed point variable. Note that for each r, 1 \leq r \leq k, there must be a statement numbered n_r, or a fatal error will result. The value of i should previously have been assigned by an ASSIGN statement.
Example: If the statement ASSIGN 47 TO INTφ was the last ASSIGN statement referring to INTφ to be executed before Gφ TO INTφ, (16, 47, 36, 9), then the latter statement will result in a transfer to statement 47.

IF ACCUMULATOR OVERFLOW n₁, n₂: results in a transfer to statement n₁ if φV (the accumulator overflow switch) is set, and to statement n₂ otherwise. However, in the normal mode of operation, accumulator overflow causes a system trap which terminates execution. It is possible to avoid this trap by using the subroutine FPTIA. The user is referred to the ILLIAC II library write-ups for details.

IF (e) n₁, n₂, n₃: results in a transfer to statement n₁, n₂ or n₃ depending whether the arithmetic expression e has a value less than, equal to, or greater than zero, respectively.

IF (SENSE LIGHT k) n₁, n₂: If sense light k (1 ≤ k ≤ 13) is on, it will be turned off, and control will transfer to statement n₁. If sense light k is off, it will remain off, and control will transfer to statement n₂. If k is > 4, then a nonfataal error message will be produced for the sake of compatibility with FORTRAN II on the 7094. k must not be a variable. The sense lights are stored in M13, which is a 13-bit modifier in fast register 7. Sense light 1 is the right-most bit of M13.

IF (SENSE SWITCH k) n₁, n₂: If sense switch k (1 ≤ k ≤ 13) has been turned on control will transfer to statement n₁, and sense switch k will remain on. If sense switch k is turned off control will transfer to statement n₂, and sense switch k will remain off. k must not be a variable.

At the moment, it is not possible to turn a sense switch on except by writing a NICAP subprogram. Some day, it may be possible to set sense switches by means of a $ SENSE SWITCH card. For those users who cannot wait, we offer the information that the 13 sense switches are stored in M12, a 13-bit register in fast register 7. Sense switch 1 is the right-most bit of M12.

If k is > 6, a nonfataal error will be generated for the sake of compatibility with FORTRAN II for the IBM 7094.
PAUSE: causes a halt order to compile. Halt is a trapped order which will
terminate the job. If a halt is desired, a \$HALT card should be used at
the start of the program (see 2.3).

PRINT \(n_p\), list: causes BCD line images for printing to be written on the output
tape. \(n_p\) must be the number of a FORMAT statement which specifies the
format of every line image. "List" is explained at the start of this
section.

PUNCH \(n_p\), list: causes BCD card images for punching to be written on the output
tape. \(n_p\) must be the number of a FORMAT statement which specifies the format
of every card image. "List" is explained at the start of this section.

READ \(n_p\), list: causes BCD card images to be read from the input tape. \(n_p\) must
be the number of a FORMAT statement which specifies the format of every card
image. "List" is explained at the start of this section. If the user attempts
to read binary card images with this statement, execution is terminated.

READ DRUM: ignored.

READ INPUT TAPE \(j, n,\) list: has the same effect as READ \(n,\) list. The tape
number \(j\) must be supplied but is ignored.

READ TAPE \(j,\) list: causes binary information to be read from one logical record
on the tape mounted on symbolic tape unit \(j\) into the locations specified in
the list. At the moment, \(j\) must equal either 6 or 7. "List" is explained
at the start of this section. A logical record is read completely only if
the list specifies as many words as the logical record contains; no more than
one logical record is read. The tape, however, always moves to the beginning
of the next logical record.

Notes: 1. A logical record is defined to be the physical records written by
a previous WRITE TAPE statement.

2. If COMMON is used in the same program as READ TAPE, 256 words at
the start of COMMON must be set aside as a buffer area (see the
explanation of the COMMON statement).
RETURN: is the last executed statement of a subprogram (see 7.1.8.2.2).

REWIND j: cause symbolic tape unit j to rewind. At the moment, j must equal either 6 or 7.

RIT j, n, list: has the same effect as READ n, list. The tape number j must be supplied but is ignored.

SENSE LIGHT k: If k = 0, this results in all thirteen sense lights being turned off. If 1 ≤ k ≤ 13, then only sense light k is turned on. If 5 ≤ k ≤ 13, then a nonfatal error is produced for the sake of compatibility with FORTRAN II for the IBM 7094. The sense lights are stored in the 13 bit modifier M13. k must not be a variable.

STOP: causes termination of execution of the program and a return of control to the ILLIAC system programs. It is identical to CALL SYSTEM as a means of terminating a job.

SUBROUTINE a(a₁, ..., aₙ): see 7.1.8.4.

WRITE DRUM: ignored.

WRITE OUTPUT TAPE j, n, list: has the same effect as PRINT n, list if j is an even number or a variable name. If j is an odd number, it has the same effect as PUNCH n, list.

WRITE TAPE j, list: causes one logical record of binary information to be written on symbolic tape unit j from the locations specified in the list. At the moment, j must equal either 6 or 7. "List" is explained at the start of this section. Note: a logical record may include several physical records but not vice versa. If COMMON is used in the same program as WRITE TAPE, 256 words at the start of COMMON must be set aside as a buffer area (see the explanation of the COMMON statement).

WOT j, n, list: is identical to WRITE OUTPUT TAPE.
7.1.8 Subprograms: Functions and Subroutines

A subprogram is a program which is used by another program (the calling program). Subprograms in general must be assembled independently of one another and independently of the calling program and then loaded into the machine all together when it comes time to run the main program. Under the present batch processor this is achieved by preceding just the complete program with an ID card and a $GÔ$ card and preceding each subprogram or calling program with a $FORTRAN$ card (or $NICAP$ etc., as the case may be). This will result in the compilation of all programs which are to be compiled followed by execution of the complete program. For further information the user is referred to Chapter 2. Note that it is possible to define a function subprogram by means of an arithmetic expression (see 7.1.8.2) and that in this case independent assembly is not required; in fact, it is not possible.

Every subprogram has a name which is assigned to it when the subprogram is defined. A subprogram is then called (i.e., used by another program) by means of its name. A detailed explanation of how to name and use a subprogram is given below, starting with Section 7.1.8.1.

As the reader may have gathered, there are two kinds of subprograms: functions and subroutines. Every function and most subroutines have associated with themselves a list of parameters. When the subprogram is defined, the list is a list of dummy parameters. Dummy parameters must be fixed or floating point nonsubscripted variables or array names. This list of dummy parameters must appear immediately to the right of the name of the subprogram when the subprogram is defined. (See 7.1.8.2 and 7.1.8.4 for methods of defining subprograms). The raison d'être of dummy parameters is that they serve as place holders in the subprogram for call parameters. Call parameters must be either fixed or floating point constants or variables, or subscripted variables, or array names, or arithmetic expressions, or subprogram names.

When the subprogram is used, a list of call parameters appears immediately to the right of the name of the subprogram. In FORTRAN, the list of call parameters and the list of dummy parameters must be of the same length, and there must be a certain amount of agreement in the characteristics of dummy and call parameters occurring at the same positions of their respective lists.
amount of agreement required is made precise below. We digress here to clarify
the notion of a placeholder: any occurrence of a dummy parameter as a place-
holder in the definition of the subprogram will be replaced for purposes of
execution by the call parameter corresponding to it when the subprogram is called.
Thus the call parameters can be used to transfer data from the calling program
to the called subprogram, and vice versa.

The agreement required between corresponding dummy and call parameters
is in the following attributes:

1. If one of the parameters is an array name, then the other one must
also be an array name. Note that a parameter is an array name if
and only if it is a variable which appears in a DIMENSION statement.
Further, the two arrays must have the same size and dimension. Note
that the size of the array to which the dummy parameter refers to
cannot depend on another dummy parameter, i.e., dynamic dimension is
not permitted.

2. If a call parameter is subscripted, then it must appear in a
DIMENSION statement in the calling program. The subscripts of a
call parameter may be constants, variables, or subscripted variables
again.

3. If a dummy parameter is used as a subprogram name in the subprogram
S, say, for which it is a dummy parameter, then the corresponding
call parameter in a call to S must appear in columns 7 - 72 of an
F card in the calling program (before it appears as a call parameter),
unless the call parameter is used as a subprogram name elsewhere in
the calling program (i.e., unless the calling program has some other
way of telling that this call parameter is actually a subprogram
name.) An F card is a card with an F in column 1. More than one
subprogram name can appear in an F card, provided that the names
appearing are separated by commas. Dummy parameters standing for
subprogram names are used in subprogram definition precisely as the
subprogram names are intended to be used. No F card is required to
identify the appearance of a dummy parameter standing for a subprogram name in a subprogram. Note that it is not permitted to call the subprogram being defined from within said subprogram, i.e., recursive definition of subprograms is forbidden.

4. No agreement is necessary with respect to parameters being fixed or floating point.

We have pointed out the use of call parameters and dummy parameters for the transfer of information between program segments. There is another method of transferring information: put it in COMMON (see 7.1.7). COMMON can only be used when the call parameters are variables, subscripted variables or array names. The advantage of using COMMON are first, that the subprogram performs fewer (if any) address constructions and second, that if COMMON is not used, then data may be transferred from the calling program to the subprogram, which takes time and space, particularly if large arrays are involved. Thus the use of COMMON means, in general, that subprograms will be executed more quickly and take less space.

A word about the distinction between subroutines and function subprograms: a function subprogram always leaves a number (the value of the function) in the accumulator when it returns control to the calling program; a subroutine does not leave anything meaningful in the accumulator. Aside from this distinction, plus the fact that a function subprogram must have a nonempty associated list of parameters, function subprograms and subroutines are the same.

Some notes:

1. Note that care must be exercised in writing subprograms to ensure that a subprogram does not change the values of the call parameters specified by the calling program before it uses them.

2. Note that if the sense light settings are changed in a subprogram, that change is effective in the main program. ML3 is used to hold the sense lights.
3. Note that if the value of a parameter in the list of dummy parameters is changed (e.g., if it appears on the left-hand side of an arithmetic statement) in the subprogram, then the corresponding call parameter must be a variable, a subscripted variable, or an array name (i.e., it should not be a constant or an arithmetic expression. If it is a constant, the value of the constant will be changed. Changing the value of an arithmetic expression in this way is meaningless).
7.1.8.1 How to Name a Function

A terminal F is allowed in a function name, and a function name can thus be up to seven characters long. Since other names in FORTRAN can only be six characters in length, it is convenient to remove the terminal F in compiling a seven-character function name. This is done. However, the terminal F is not removed from a function name of six or fewer characters in length (except for library functions—see 7.1.8.2). Thus the labels NAMEF and NAME will be distinguished by FORTRAN but NAMINGF and NAMING will not be distinguished.

In general, the terminal F is not required on the names of user defined functions in this version of FORTRAN, although it may be used. If, however, a function name of four or more characters in length ending in F is used, then it refers to a fixed or floating point valued function depending on whether the first letter of the name is X or not. If the name is fewer than four characters or if it does not end in F, then it is fixed or floating valued according as it begins with one of I, J, K, L, M or N or not. Thus XPRTF, INF, and INTCGER are fixed point valued; XPRT, IRSF, PAD, and XAF are floating point valued.
7.1.8.2 How to Define a Function

Functions are defined in three different ways:

1. Library functions and built-in functions (all built-in functions and some library functions are predefined in FORTRAN).

2. Arithmetic statement functions

3. Function subprograms

FORTRAN provides the following library functions for the user:

\[
\begin{align*}
\text{SQRTF}(\alpha) &= \sqrt{\alpha} \\
\text{ELGOF}(\alpha) &= \log_e \alpha \\
\text{BLGOF}(\alpha) &= \log_2 \alpha \\
\text{TLOGF}(\alpha) &= \log_{10} \alpha \\
\text{EXPF}(\alpha) &= e^\alpha \\
\text{SINF}(\alpha) &= \sin(\alpha) \quad (\alpha \text{ in radians}) \\
\text{COSF}(\alpha) &= \cos(\alpha) \quad (\alpha \text{ in radians}) \\
\text{TANHF}(\alpha) &= \tanh(\alpha) \quad (\alpha \text{ in radians}) \\
\text{ATANF}(\alpha) &= \arctan(\alpha) \quad \text{radians}
\end{align*}
\]

Note: The final F is optional in all of the above names; thus \text{SIN}(\alpha) means the same as \text{SINF}(\alpha).

In addition, the following functions are "built-in" to FORTRAN (\alpha and \beta should be floating point unless otherwise specified):

\[
\begin{align*}
\text{ABSF}(\alpha) &= |\alpha| \\
\text{XABSF}(\alpha) &= |
\begin{cases} 
\alpha & \text{if } \beta \geq 0 \\
-\alpha & \text{if } \beta < 0 
\end{cases}
\end{align*}
\]
FL\textsuperscript{TF}(\alpha) = \alpha \text{ (change from fixed to floating)} \quad \text{Not necessary on ILLIAC II}

XFIXF(\alpha) = \alpha \text{ (change from floating to fixed)}

DIMF(\alpha,\beta) = \alpha - \text{MIN}(\alpha,\beta)

XDIMF

MAXOF(\alpha,\beta)\textsuperscript{2} = \text{maximum of } \alpha \text{ and } \beta \text{ where } \alpha \text{ and } \beta \text{ are fixed point}

XMAXOF\textsuperscript{3}

MAXIF(\alpha,\beta)\textsuperscript{2} = \text{maximum of } \alpha \text{ and } \beta \text{ where } \alpha \text{ and } \beta \text{ are floating point}

XMAXIF\textsuperscript{3}

MINOF\textsuperscript{3}

XMINOF\textsuperscript{3}

MINIF\textsuperscript{3}

XMINIF\textsuperscript{3}

defined by analogy with MAX.

Notes:

(1) X prefixed indicates that the value of the function is fixed point rather than floating.

(2) \([\alpha] = \text{greatest integer } \leq \alpha.\)

(3) These functions are restricted to two arguments.

(4) The final F must be present on all built-in function names.

The distinctions between library and built-in functions are minimal. First, the user may, with the approval of the system programming group, add to the set of library functions (see 7.1.8.2.3) but not to the set of built-in functions; second, the compiler will not have to search the library tape to identify built-in functions (although it does at this time); and third, some built-in functions (such as MAX\textsuperscript{TF}) will eventually have a variable number of arguments.

The user may define functions himself in three different ways which are described in the next three sections.
7.1.8.2.1 Arithmetic Statement Functions

This type of function is simply defined within any FORTRAN program or subprogram by setting the function name, followed by brackets enclosing the dummy parameter list, equal to the desired arithmetic expression.

Examples:

1. The statement
   \[ F(X,Y) = (X \times Y - X \times X^2) / (X + Y + 3.0) \]
   defines a function \( F \) of two variables; the statement
   \[ G(U,V,W) = F(SIN(U \times X^2 + V \times X^2), W) + F(COS(U \times X^2 + V \times X^2), W) \]
   uses \( F \) to define a new function, \( G \), of three variables.

2. Subscripted variables are not allowed as dummy parameters in this type of function definition. Thus the statements
   \[ F_1(X) = X(10) \]
   \[ F_2(X) = X(1) \]
   \[ F_3(X,J) = X(J) \]
   are illegal, but
   \[ F_4(I) = Y(I)^2 \]
   is legal.

Note that no function can call itself directly or indirectly in its definition. Thus for arithmetically defined functions, the following examples are grossly illegal:

1. \( F(0) = 1 \)
   \[ F(X) = F(X - 1)^2 \times X \]

2. \( G(0) = 0 \)
   \[ F(0) = 1 \]
   \[ F(X) = G(X - 1) + X \]
   \[ G(X) = F(X - 1)^2 \times X \]
Arithmetic statement functions are local to the program or subprogram in which they are defined, with one exception: the name of such a function may appear as a call parameter in its defining program, in which case the function may be used by the called subprogram.
FUNCTION \( a(a_1, \ldots, a_n) \)

Here \( a \) is a fixed or floating point name and \((a_1, \ldots, a_n)\) is a list of dummy parameters. The name of the function, \( a \), must obey the rules for function names given in 7.1.8.1. The statement FUNCTION \( a(a_1, \ldots, a_n) \) is then the first statement of a function subprogram (although this is not checked). The last statement executed by the function subprogram must be RETURN or END. The variable \( a \) must occur at least once on the left hand side of an arithmetic expression or in an input statement list in the function subprogram. Aside from these restrictions, the function subprogram may be any legal FORTRAN program.

It is also possible to encode function subprograms in NICAIF if the user wants some portion of his object program to be more efficient than an object program written in the FORTRAN source language. For this purpose, the user needs to know the following facts:

1. The FORTRAN compiler computes all subscripts occurring in the list of call parameters, and then compiles JSB 3, FIL, followed by as many DECQ's as are needed to give the addresses of the call parameters in the same order as specified by the calling program. A FIL is generated after the last parameter. If the parameter in a call statement is a subprogram name, or an indexed variable, the parameter given in the DECQ sequence is the address of a temporary cell containing the transfer vector or the value of the variable respectively. If the parameter is a number, then the DECQ parameter is the address of a cell containing that number in floating point.

2. Fast registers F4, F5 and F6 must be saved (the indices of D0's and other such valuable information is stored therein).

3. M12 and M13 contain the sense switches and lights.

4. The resulting value of the function must be stored in the accumulator before returning.
7.1.8.2.3 User Defined Library Functions

The user may redefine by a SUBROUTINE or FUNCTION subprogram any library program. He should, however, beware of the names PRINT, READ, PUNCH, I$LIST and any name starting with SYS-- which are used by I/O statements since if he uses these names, the program so named will no longer be available to him.

Example: The subprogram

```
SUBROUTINE PRINT(X)
  PRINT 1,X
  1 FORMAT (1H, F10)
END
```

will cause the name PRINT to be redefined. In fact, the program above causes an infinite loop to assemble, since the statement PRINT 1,X in it assembles as a call to it.
7.1.8.3 How to Name a Subroutine

Any fixed or floating point variable can be used as the name of a subroutine except as mentioned in 7.1.8.2.3.
7.1.8.4 How to Define a Subroutine

**FORTRAN** does not provide any built-in subroutines. Subroutines are defined by writing a subprogram which begins with `SUBROUTINE a(a_1 \ldots a_n)`, (a is the name of the subroutine; (a_1 \ldots a_n) is the list of dummy parameters: it may be empty) and which consists of any legal sequence of **FORTRAN** statements such that the last executed statement is always RETURN or END. It is not necessary that the variable a appear on the left hand side of an equation or that it appear in an input list.

Subroutines may also be encoded in NICAP as for function subprograms except that nothing need be returned to the accumulator.
7.1.8.5 How to Use a Subprogram

The list of call parameters and the use of COMMON have been explained in 7.1.8 and 7.1.7. It remains to explain the CALL statement and give examples.

CALL may be used to call any subprogram. This is not usually the case in FORTRAN.

Suppose that the subprogram name and list of dummy variables not specified in COMMON are \( a(a_1 \ldots a_n) \). Then to call the subprogram \( a \) with call parameters \( (b_1 \ldots b_n) \), one writes the statement CALL \( a(b_1 \ldots b_n) \) in the calling program. This will result in automatic transfer to, execution of, and return from the subprogram using the parameters \( b_1 \ldots b_n \) in place of \( a_1 \ldots a_n \). Parameters specified in COMMON may appear as call parameters, although they normally do not. Note however, that it does not make sense in general for parameters specified in COMMON to appear as dummy parameters, since it results in the contents of COMMON being overwritten.

To understand the effect of doing so the user should understand the way in which data is transferred. Separate storage is allocated to variables in subprograms. (This is the reason for using COMMON where possible.) On entry to the subprogram, all of the data indicated in the CALL list is copied from the calling program storage area into the subprogram area. Thus if the subprogram has dummy variables in COMMON, they will be overwritten at this time. On return from the subprogram, the data is copied back from the subprogram storage area to the main program area. If any of the call parameters are in COMMON, they may be overwritten at this point.

A function subprogram may also be used without using the CALL statement. (So may subroutines: again, this is unusual FORTRAN.) In fact any mention of the name of a function (together with a correct list of call parameters) in an arithmetic expression will result in execution of the function subprogram.

\[
\sum_{i=1}^{N} X(i)
\]

Example: The following function subprogram computes \( \sum_{i=1}^{N} X(i) \) for values of \( N \leq 20 \):
FUNCTION PWRPLSF (X,Z,N)
DIMENSION X(20)
Y = 0.0
DO 1 I = 1,N
1 Y = Y + X(I)
PWRPLSF = Z ** Y
END

The following program uses this function subprogram to compute and print successively the quantities

\[ \sum_{i=1}^{k} i, \quad k = 1, 2, \ldots, 15: \]

DIMENSION A (20)
B = 2.0
DO 10 K = 1,15
A(K) = K
X = PWRPLSF(A,B,K)
10 PRINT 12,X
12 FORMAT (1HO, F30)
END
The output from this program is as follows:

2
8
64
1024
32768
2097152
268435456
68719476736
351843720888484
36028797018533888
73786976293932236800
302231454899379506249728
2475880078526850453431386061
+.405648192066669281773515E+32
+.132922799575000813986243E+37

This output appears to be most impressive; unfortunately, however, only the 12 most significant digits have meaning in these numbers. The remaining digits are garbage.

Note the following point to beware of: suppose we define

SUBROUTINE SET1(S)
X = 1
RETURN

Then if we write the statement

CALL SET1 (2),

this will result in the location which contained the value 2 being changed to contain the value 1.
7.1.9 Printed Output From a Compilation

If a $PRINT OBJECT control card is present at the front of the program deck to be compiled, the following items appear on the listing in the order given (these items are explained below):

(a) ID information, date, control cards
(b) FORTRAN source program listing
(c) Error messages, if any; these are self-explanatory
(d) Compiled object program, in machine language (it is hoped eventually to print a NICAP version of the object program beside the machine language version)
(e) LOCATIONS OF VARIABLES NOT APPEARING IN FUNCTIONS OR DIMENSION STATEMENTS
(f) LOCATIONS OF DIMENSIONED VARIABLES AND FUNCTION NAMES
(g) LOCATIONS OF STATEMENT NUMBERS USED BY THE SOURCE PROGRAM.

If there is no $PRINT OBJECT control card, item (d) will not appear, but everything else will. Further, if a $G0 control card is present, then item (h) will appear, independently of the presence of the $PRINT OBJECT card:

(h) MEMORY MAP.

The following explanation is offered of items (a) through (h) (this explanation is meant to be read in connection with a listing of a FORTRAN compilation).

(a) First line: The left-most item is a sequence number indicating what batch, and where in the batch, the job was run. The remaining items are self-explanatory.
Second line: The left-most portion is a copy of the ID card. The remaining three items are the date, the sequence number (again), and the time of day at which the job was started, in hours, minutes, seconds, and decimal points of a second.
Third line, and possibly fourth line, fifth line, etc. These lines are copies of the control cards.

Date: 6/3/65
Section: 7.1.9
Page: 1 of 4
Change: ILLIAC II MANUAL
Last line: This specifies the version of FORTRAN used to compile the program.

(b) Self-explanatory (Note that fatal error messages about undefined statement numbers print out immediately following item (b), and are not included in item (c)).

(c) SEQUENTIAL STATEMENT NUMBER BELOW: Refers to the sequence numbers assigned by the compiler to each statement. These sequence numbers appear to the left of the statement as listed in (b). Note that comment cards are not given statement numbers. Note also that the word FATAL will appear to the right of statement numbers referring to fatal errors; if the error is nonfatal, nothing is printed to the right of the statement number.

(d) The reader is referred to Chapters 2, 3, and 4 of the present manual for an explanation of ILLIAC II machine language. We content ourselves here with the observations that the left-most column of figures consists of the decimal numbers assigned to successive instructions in the NICAP program generated by the compiler from the user's FORTRAN program; that the second column consists of the relocatable decimal storage locations assigned to this program by the NICAP assembler; that the third column is a translation of the second into octal; and that the machine language listing itself is in octal. Note also that the first few words of the program contain transfer vectors to subprograms, if any of the latter have been used.
(e) 1. The name of a function appears here on the listing of the function subprogram. The location given is the temporary storage location of the value of the function within the subprogram.

2. Internal names used by the compiler and having no significance for the FORTRAN user also appear here. These names may be identified by the fact that they start with a number while all of the user's names start with a letter.

(f) 1. The name of a function or subroutine subprogram appears here on the listing of the subprogram. The location given is the entry point to the subprogram. (Note: The entry to a subprogram is near the end, not the beginning, of the listing of the subprogram, i.e., the entry to a subprogram is generally nearer to the highest core location used by the subprogram than to the lowest.)

2. The names of subprograms called in the program being listed appear here. The location given is the location of the transfer vector (to the subprogram) within the program being listed.

(g) Each statement number appearing in the user's program appears here at least once, independently of whether any reference is made to it by statements in the user's program. Normally, the octal address given for each statement number is the quarter word to which the statement number refers. However, statement numbers appearing in DO statements (e.g., 13 appears in DO 13 I = 1, 33, 2) may appear here more than once in which case the situation is more complicated.

To be precise, a statement number will appear here once for each DO statement that it appears in the user's program; the octal address given for each DO statement appearance will be the location of the start of the DO loop to which the DO statement refers. Further, if a reference is made
in a transfer statement (e.g., in an IF statement or in a \( \text{IF} \space T \) statement) to a statement number appearing in a \( \text{DO} \) statement, then that statement number will appear here once more than the number of \( \text{DO} \) statements in which it appears, and the octal address given in this case will be the location of the first machine language instruction in the sequence of machine language instructions used to close the \( \text{DO} \) loop (this address will be greater than any of the addresses associated with appearances of the statement number in a \( \text{DO} \) statement).

(h) 1. **COMMON**: location given is the start of the **COMMON** area of core.

2. **ERASABLE**: location given is the start of the **ERASABLE** area of core.

3. **AVAILABLE MEMORY STARTS AT**: location given is the highest location used by the program, except that the monitor is, of course, always in locations 16,000\(_8\) through 17,777\(_8\) (the four high blocks of core).

4. **PROGRAM EXECUTION BEGINS AT**: location given is the absolute address of the first executable instruction in the object program; in order to find out how much the program has been relocated by, it is necessary to find out how many transfer vectors there are, and subtract that number from the location given.

5. **LOCATION OF SUBROUTINES USED**: location given is the entry point to the subroutine. The relocation of the subroutine can be computed by subtracting the location mentioned in (f)1. from the location given here.

Note: No indication is given when a program exceeds the amount of core available to it; the excess only becomes apparent when the program is executed.
CHAPTER 8. THE PROGRAM LIBRARY

TABLE OF CONTENTS

8.1 Introduction
8.2 Classification
8.3 Subroutine Conventions
8.4 Program Descriptions

<table>
<thead>
<tr>
<th>Change</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7/10/64</td>
</tr>
<tr>
<td>2</td>
<td>11/14/64</td>
</tr>
<tr>
<td>3/05/63</td>
<td></td>
</tr>
</tbody>
</table>

8.4-B1-ATAN1 7/20/64
8.4-B1-SIN1   7/20/64
8.4-B3-COSH1  7/29/64
8.4-B3-EXP1   7/29/64
8.4-B3-LGT1   11/26/63
8.4-B3-SINH1  7/29/64
8.4-B4-SQR1   10/10/63
8.4-D1-GQU1   11/19/63
8.4-D2-RKQ1   7/15/64
8.4-EL-DVDF1  7/20/64
8.4-EL-LAG6   11/15/63
8.4-EL-LGUN   11/15/63
8.4-F4-SLQ1   7/20/64
8.4-G5-RAN1   8/10/64
8.4-J6-TOPS   11/14/64
8.4-K0-IPLIST 7/20/64
8.4-M0-CMPL   9/22/64
8.4-M2-PRINT  7/16/64
8. THE PROGRAM LIBRARY

8.1 Introduction

Programs in the library fall into three classes:
1) subroutines
2) complete programs not of the system type
3) system programs (e.g., assemblers, compilers, general I/O programs, monitors).

This chapter will describe classes 1) and 2) completely and the programming details of class 3). All of class 3) will be on tape unless their use has been discontinued. (Later on, disc files will be used instead of tape.) Some of classes 1) and 2) will be on tape (as many as possible). The remainder will be available on cards.
8.2 Classification

The numbering of the program is as follows. The number will be made up of five parts,

CC, MMM...M, LL, QQ, SS

where

CC is a classification code identical to the one used in the SHARE 7090 library. (See Digital Computer Laboratory Technical Progress Report, April, 1962, pp. 42-46 for details.)

MMM...M is a four- to six-letter semi-mnemonic identification of the routine. It is unique to each routine.

LL is the change level. If the description is changed in any way, the change number will be incremented. Normally the routine itself should not be changed.

QQ is the area of origination, normally UI.

SS is a code specifying the system and/or language in which the program is available. No assignments have been made to this yet.
8.3 Subroutine Conventions

Subroutines in the library follow the following conventions.

ENTRY Made with a JSB3, ....

A subroutine may start in any control group unless the description states otherwise. If a subroutine does have requirements on the quarter word placing, it will take care of them automatically by FIL pseudo operations.

EXIT

Made from subroutine by JLB3. Therefore the entry JSB3, ... must be followed by a FIL. This is taken care of by using the CALL pseudo operation.

Parameters and Data

ON ENTRY

Thirteen-bit parameters or addresses are handled as follows:

Single address length parameter in M1. More than four are packed four per word in consecutive locations in memory called control words. The address of the first location is in M1. Less than four but more than one are packed into the word followed the location of the address of the JSB3,... instruction used for entry.*

Fifty-two bit parameters or data appear in memory with the locations of individual words or the start and extent of blocks as 13-bit parameters. When there are one or two special data words, they may be in the Accumulator if only one, or the Accumulator and F2 in the case of two words.

* Some parameters fit more naturally into control words, some into the word(s) following the entry jump order, so this convention may be dropped as experience is gained in the use of this machine.
ON EXIT

A single 13-bit answer is put in M0. When there is more than one 13-bit answer, they are packed, four per word, into control words. Fifty-two bit answers are put into locations specified in control words on entry. One or two words of answer may appear in the accumulator and F2. Exit is made to the left-hand control group in location M3 except if parameters appeared in the order stream after the JSB3, ... entry, in which case exit is made to the left-hand control group of the first location free of parameters.

TEMPORARY STORAGE

Subroutines which are complete in the sense that they do not use user-supplied subroutines as auxiliary subroutines use locations COMMON, COMMON+1, ..., etc., as far as necessary.

Subroutines on the library tape are in binary, and therefore do not require the programmer to define COMMON. However, if the NICAF deck is used, COMMON must be defined by the programmer.

Subroutines using auxiliary subroutines require that M2 contain a location which is the start of a block of free locations of sufficient length. For example, the Runge-Kutta integration of ordinary differential equations uses four temporary storage locations. If M2 = 100 on entry to Runge-Kutta, these locations are 100, 101, 102 and 103. On entry to the auxiliary subroutine, M2 will contain 104; on return to the main program, it will contain 100 again.
NAME: Arctangent Subroutine

PURPOSE: Computes arctangents of arguments

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: Four words beginning with COMMON

NUMBER OF WORDS: 51 words

EXECUTION TIME: 210 μsec (average)

USE: Normal entry with argument in accumulator. Normal exit with result in accumulator. F2 through F7 saved.

METHOD: The sign of the argument is noted and the absolute value used to compute the arctangent. The correct sign is restored before exit. To obtain the arctangent, the domain of X is divided into seven intervals as follows:

<table>
<thead>
<tr>
<th>No.</th>
<th>Interval</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$0 \leq X &lt; \tan \frac{\pi}{24}$</td>
<td>Use $P(x)$ directly</td>
</tr>
<tr>
<td>1</td>
<td>$\tan \frac{\pi}{24} \leq X &lt; \tan \frac{3\pi}{24}$</td>
<td>Use Eqn A with $K = 1$</td>
</tr>
<tr>
<td>2</td>
<td>$\tan \frac{3\pi}{24} \leq X &lt; \tan \frac{5\pi}{24}$</td>
<td>Use Eqn A with $K = 2$</td>
</tr>
<tr>
<td>3</td>
<td>$\tan \frac{5\pi}{24} \leq X &lt; \tan \frac{7\pi}{24}$</td>
<td>Use Eqn A with $K = 3$</td>
</tr>
<tr>
<td>4</td>
<td>$\tan \frac{7\pi}{24} \leq X &lt; \tan \frac{9\pi}{24}$</td>
<td>Use Eqn A with $K = 4$</td>
</tr>
<tr>
<td>5</td>
<td>$\tan \frac{9\pi}{24} \leq X &lt; \tan \frac{11\pi}{24}$</td>
<td>Use Eqn A with $K = 5$</td>
</tr>
<tr>
<td>6</td>
<td>$\tan \frac{11\pi}{24} \leq X &lt; \infty$</td>
<td>Use Eqn B</td>
</tr>
</tbody>
</table>

Programmed by: John Kelly
Approved by: [Signature]

Date: 7/20/64
Section: 8.4-Bl-ATAN1
Page: 1 of 5
Change: [Blank]
METHOD (Continued):

Eqn A: $\arctan X = \frac{k\pi}{12} + \arctan t_k$, $t_k = \frac{X - \tan \frac{k\pi}{12}}{1 + X \tan \frac{k\pi}{12}}$

Eqn B: $\arctan X = \frac{\pi}{2} - \arctan \frac{1}{X}$

$P(x)$ is used to compute $\arctan t_k$ and $\arctan \frac{1}{X}$.

$P(x) = a_1x + a_3x^3 + a_5x^5 + \ldots + a_{17}x^{17}$

where

$a_1 = 1.000000 00000 00000$
$a_3 = -0.33333 33333 33333$
$a_5 = 0.199999 99999 99998$
$a_7 = -0.14285 71428 56331$
$a_9 = 0.11111 11111 07794$
$a_{11} = -0.09090 90909 63368$
$a_{13} = 0.07692 04073 24915$
$a_{15} = -0.06666 66667 41311$
$a_{17} = 0.05467 21009 39594$

ACCURACY:
12 decimal digits

Maximum error 0-1.0 .23 $10^{-12}$
Average absolute error 0-1.0 .5 $10^{-13}$

Maximum error 0-0.1 .7 $10^{-13}$
Average absolute error 0-0.1 .11 $10^{-13}$

REFERENCE:
ATAN1 SFR 5, COMMON
    SAVE F5.
SFR 2, COMMON+2
    SAVE F2.
SFR 3, COMMON+3
    SAVE F3.
TN ATAN1G
    WANT ABSOLUTE VALUE OF X.
STR F2
    SAVE X.
CAM 4
    SET POSITIVE SIGN FLAG.
ATAN1A CNM 5
    SET M5 FOR INTERVAL FLAG.
ATAN1B SUB ATAN1K+M5+1
    FORM NEXT DIFFERENCE TO GET INTERVAL.
ADM 5, 1
    INCREMENT INTERVAL FLAG.
T2P ATAN1B
    JUMP IF INTERVAL NOT YET REACHED.
JZM 5, ATAN1C
    JUMP IF INTERVAL 0 TO EXECUTION OF POLY.
CAM 6, M5-6
    TEST FOR INTERVAL 6.
JZM 6, ATAN1H
    JUMP IF INTERVAL 6.
CAD F2
    ENTER X.
SUB ATAN1L+M5-1
    FORM X - TAN(K*PI/12).
STR F3
    STORE NUMERATOR X - TAN(K*PI/12).
ADD 1.
    1 + X * TAN(K*PI/12).
VID F3
    T(K)=(X-TAN(K*PI/12))/(1+X*TAN(K*PI/12)) ATAN1 20
SFR 6, COMMON+1
    SAVE F6.
LFR 6, ATAN1M+M5-1
    STORE K*PI/12 IN F6.
STR F2
    SAVE T(K) IN F2.
ATAN1C CAD F2
    ENTER X, T(K), OR 1/X. CALL IT Y.
MPY F1
    FORM Y**2.
CAM 6, ATAN1J
    SET LOCATION OF FIRST COEFFICIENT.
CSM 7, 8
    SET POLY END TEST COUNTER.
STR F3
    SAVE Y**2 IN F3.
CAD 6, 1
    ENTER FIRST COEFFICIENT A(17).
FLD
    POLY MEANS THE POLYNOMIAL EXPRESSION.
MPY F3
    FORM POLY * Y**2.
ADD 6, 1
    FORM POLY + NEXT COEFFICIENT.
CJF  7,  POLY ENDTEST.
MPY  F2  POLY=A(1)*Y+A(3)*Y**3+...+A(17)*Y**17.
JZM  5,ATAN1D  JUMP IF INTERVAL 0.
JNM  5,ATAN1I  JUMP IF INTERVAL 6.
ADD  F6  ARCTAN(X) = POLY + K*PI/12.
ATAN1D JZM 4,ATAN1E  JUMP IF POSITIVE SIGN FLAG WAS SET.
STN  F0  X WAS NEGATIVE, SO MAKE
CAD  F0  ARCTAN(X) NEGATIVE.
ATAN1E JZM  5,ATAN1F  JUMP IF INTERVAL 0.
LFR 6,COMMON+1  RESTORE F6.
ATAN1F LFR  5,COMMON  RESTORE F5.
LFR  2,COMMON+2  RESTORE F2.
LFR  3,COMMON+3  RESTORE F3.
JLH  M3  EXIT ATAN1 SUBROUTINE.
ATAN1G STN  F2  SAVE ABSOLUTE X.
CNM  4  SET NEGATIVE SIGN FLAG.
CAD  F2  ENTER X = ABSOLUTE X.
TRA  ATAN1A  JUMP TO INTERVAL TEST.
ATAN1H CAD  1.  SET NUMERATOR.
DIV  F2  FORM Y = 1/X.
SFR  6,COMMON+1  SAVE F6.
CNM  5  SET INTERVAL MODIFIER M5 NEGATIVE.
LFR  6,ATAN1N  STORE PI/2 IN F6.
STR  F2  SAVE 1/X IN F2.
TRA  ATAN1C  JUMP TO EXECUTE POLY.
ATAN1I SUB  F6  -ARCTAN(X) = ARCTAN(1/X) - PI/2.
ADM  4,1  REVERSE SIGN SINCE -ARCTAN(X) < ZERO.
TRA  ATAN1D  JUMP TO EXIT.
FIL  THE FOLLOWING ARE THE TABLES USED.
ATAN1J OCTQ 6776, 17664, 13050, 1776, 15676, 725, 6430, 4777 A(17), A(15).
OCTQ 2354, 4162, 4733, 377, 15056, 10570, 6100, 11377 A(13), A(11).
OCTQ 3434, 7070, 15622, 3777, 13333, 6666, 15556, 11777 A(9), A(7).
OCTQ 6314, 14631, 11463, 3377, 15252, 12525, 5252, 12600 A(5), A(3).
OCTQ 2000, 1,  A(1).
ATAN1K OCTQ 4154, 17723, 1623, 1577, 2205, 5730, 430, 2400 TABLE OF
OCTQ 2646, 5510, 6660, 5600, 4223, 505, 16475, 11600 DIFFERENCES
OCTQ 2161, 12334, 6504, 10001, 2456, 7456, 2032, 12202 FOR
OCTQ 7777, 17777, 17777, 17677 THE INTERVAL TEST.

ATAN1L OCTQ 2111, 10242, 17236, 14600, 4474, 15164, 5440, 6400 TABLE
OCTQ 2000, 1, 3355, 11727, 4130, 4601 OF
OCTQ 7355, 11727, 4130, 4601 TAN(K*PI/12).

ATAN1M OCTQ 2060, 5221, 14055, 7200, 4140, 12443, 10132, 16400 TABLE
OCTQ 6220, 17665, 4210, 5600, 2060, 5221, 14055, 7201 OF
OCTQ 2474, 6466, 3070, 15001 K*PI/12.

ATAN1N OCTQ 3110, 7732, 12104, 2601 PI/2.
NAME: Sine-Cosine

PURPOSE: To compute $\sin \frac{x}{\pi}$ or $\cos \frac{x}{\pi}$ with $x$ in the accumulator.

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: One location at COMMON, to be defined by the programmer and F2.

NUMBER OF WORDS: 13 words

FAST REGISTERS CHANGED: F2 and the calling sequence uses M3, hence F4.

EXECUTION TIME: 65 $\mu$sec (November, 1963)

EXIT: Normal to left hand first word with $\sin \frac{x}{\pi}$ or $\cos \frac{x}{\pi}$ in accumulator.

USE: Normal, i.e., CALL SIN1, CALL COS1

METHOD: Chebyshev Polynomials

Entrance at "SIN1" places $x - 1/2$ in A and computes $\cos \pi(x - 1/2) = \sin \pi x$. $x(\mod 2)$ is formed at "COS1" and then $x = 1/2 - |x|$. The identity $\sin(\pi/2 - |\pi x|) = \cos \pi x$ is used at this point.

Sin $\pi x$ is now computed using the Chebyshev polynomial approximation of degree 13 to the Taylor series expansion of $\sin \pi x$, $-1/2 \leq x \leq 1/2$. The polynomial

$$
\sin x = \sum_{k=0}^{6} \frac{x^{2k+1} C_k}{2k+1}
$$
METHOD (Continued): is evaluated by the standard technique. The coefficients were calculated on ILLIAC II, starting with

$$\sin y \sim y - \frac{y^3}{3!} + \ldots - \frac{y^{19}}{19!}$$

FLD
DECQ -6, COS1+5
MODIFIER CONSTANTS
SIN1
SUB 10,3,2048
COMPUTE SINE
COS1
STF 0,3,
SFR 6, COMMON
TOR 1, COS1+1
SAVE F6
DAV 10,3,2048
CLEAR OV
STN F2
1X1 - 1/2 = A
MPY F0
X = 1/2 - 1X1
LFR 6, COS1-1
SET M8 AND M9, I = -6
STR F0
STORE X**2 IN F0
CAD 9,1,
C7 = A
MPY F0
*C**2
ADD 9,1,
+C-1
ADD 9,1,
+ C-1
CJF 8,1,
IS I =0
LFR 6, COMMON
RESTORE F6
MPY F2
*X
JLH 3,**
= EXIT
OCTQ 3517,6174,4521,15173
C7
OCTQ 14165,15472,10711,10575
C6
OCTQ 2501,15532,14354,1577
C5
OCTQ 13151,6467,17623,6400
C4
OCTQ 5063,5706,6336,13601
C3
OCTQ 15325,1030,14735,4602
C2
OCTQ 6220,17665,4210,14201
C1
COMMON BSS 1
NAME: Hyperbolic cosine

OTHER SUBROUTINES USED: EXPl

TEMPORARY STORAGE: None

NUMBER OF WORDS: 5 words

EXECUTION TIME: 161.4 μsec (average \(0 \leq x \leq 2\))

USE: Standard CALL COSHl (x in the accumulator) and normal exit (cosh x in the accumulator). Fast registers are saved. OV is cleared.

MATHEMATICAL METHOD: \(\cosh x = \frac{1}{2} (e^x + 1/e^x)\) where \(e^x\) is computed by EXPl.

ACCURACY: Using the identity \(\cosh(2x) = 2 \cosh^2 x - 1\), the maximum relative error is \(7.9 \times 10^{-12}\) and the average error is \(1.0 \times 10^{-12}\).
<table>
<thead>
<tr>
<th>COSH1</th>
<th>SFR</th>
<th>4, COSH1A</th>
</tr>
</thead>
<tbody>
<tr>
<td>CALL</td>
<td>EXP1</td>
<td></td>
</tr>
<tr>
<td>STR</td>
<td>F0</td>
<td></td>
</tr>
<tr>
<td>VID</td>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>F0</td>
<td></td>
</tr>
<tr>
<td>MPY</td>
<td>10,3,2048</td>
<td>COSH1(X) IS</td>
</tr>
<tr>
<td>LFR</td>
<td>4, COSH1A</td>
<td>COMPUTED</td>
</tr>
<tr>
<td>JLH</td>
<td>M3</td>
<td></td>
</tr>
<tr>
<td>COSH1A BSS</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

SAVE F4.
EXP1 SUBROUTINE.
COSH1(X) = (1/2) * (E**X + E**(-X)).
EXIT COSH1 SUBROUTINE.
COSH1 TEMP STORAGE.
NAME: Exponential

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: None

NUMBER OF WORDS: 40 words

EXECUTION TIME: 127.5 μsec

USE: Standard, CALL EXPL (X in the accumulator) and normal exit (e^X in the accumulator). Fast registers are saved. OV is cleared but will be set if x ≥ 127 log 2 [≈ 88*].

MATHEMATICAL METHOD: \[ e^x = K^w \text{ where } K = 2^{1/8}, w = x \log_b e. w = 16a + b + f \]

where \( 0 \leq b \leq 15 \) (b integer), \( 0 \leq f < 1 \). So \[ e^x = 4^a \cdot K^b \cdot K^f. \]

a is placed in the exponent register and \( K^b \) comes from a table look-up. \( K^f = e^{cf} \) where \( c = \ln K. \)

\[ e^z = 2^{\left(\frac{1}{2} + \frac{z}{F(z)}\right)} \]

where

\[ F(z) = 2 - z + \frac{1}{6} z^2 - \frac{1}{360} z^4 + \frac{1}{15120} z^6 + \ldots \]

So

\[ e^{cf} = 2^{\left(\frac{1}{2} + \frac{f}{2 - f + \frac{c}{6} \cdot f^2 - \frac{c^3}{360} f^4 + \frac{c^5}{15120} f^6}\right)} \]

*There is a correct 8-bit exponent in the accumulator. This may be too large to store.*

Programmed by: N. T. Hamilton
Approved by: [Signature]

Date: 7/29/64
Section: 8.4-B3-EXPL
Page: 1 of 4
Change:
ACCURACY: Using the identity $e^x = 1/e^{-x}$, the maximum relative error is $3.9 \times 10^{-13}$. 
EXP1
TOR 2, EXP1
MPY EXP1F
SFR 4, EXP1E
SFR 5, EXP1E+1
SAM F5
LIN 8
SEQ F0
SAM F4
TNOR EXP1C
JNM 4, EXP1B
EXP1A
DIV 15, 3
EXP1B
CAD 15, 3
LFR 4, EXP1E
LFR 5, EXP1E+1
JLH M3
EXP1C
AND EXP1G
STR F5
MPY F0
CAM 2, EXP1H
STR F0
MPY 2, 1
ADD 2, 1
CRM 1, 9
ADD 2, 1
MPY F0
ANM 1, 15
ADD M2
SUB F5
VID F5
ANN 8064
CAM 2
ORM 3, 127
CNM 3
JZM 2, EXP1D

CLEAR OV.
FORM W = X * LOG2E. E**X = K**W.
SAVE F4
AND F5.
W = 16*A + B + F.
SHIFT AND
STORE A IN MO
AND B IN M1.
JUMP IF W NOT TOO EXTREME.
JUMP TO SET UF IF X NEGATIVE.
SET OV.
CLEAR ACC OR SET UF.
RESTORE F4
AND F5.
EXIT EXP1 SUBROUTINE.
MASK W TO LEAVE FRACTIONAL
PART F AND SAVE IN F5.
FORM F**2.
SET LOCATION OF POLY COEFFICIENTS.
SAVE F**2.
FORM THE POLY:
EXPANSION FOR
(1/2) * K**F, K = 2**(1/8).
RIGHT JUSTIFY B.
FORM
POLY.
MASK M1 TO LEAVE B.
CONTINUE
FORMING
POLY.
TEST FOR OV. LEFT 6 BITS
MUST BE 0 TO PASS.
TEST FOR UF. LEFT 6 BITS
MUST BE 1 TO PASS.
JUMP IF NO OV.
JZM 3,EXP1D
JPM 1,EXP1A
TRA EXP1B
EXPI D
ADD 10,3,2048
CAE MO
MPY M1+EXP1I
TRA 1,EXP1B
EXP1E
BSS 2
EXP1F
OCTQ 05612,12166,05127,00202
EXP1G
OCTQ 00000,00777,17777,17600
EXP1H
OCTQ 00000,00000,00260,03001
EXPI I
OCTQ 17777,17760,15377,06401
EXPIJ
OCTQ 00016,14460,03775,04201
EXPIK
OCTQ 02705,05073,02453,10203
EXPL
OCTQ 04000,00000,00000,00001
EXPM
OCTQ 04271,05603,14372,12001
EXPR
OCTQ 04603,07740,12143,07001
EXPS
OCTQ 05137,16655,05154,05001
EXPT
OCTQ 05520,04746,06376,07401
EXPU
OCTQ 06126,07124,02125,04001
EXPV
OCTQ 06564,04771,11265,12401
EXPW
OCTQ 07254,00615,14767,04401
EXPX
OCTQ 02000,00000,00000,00002
EXPY
OCTQ 02134,12701,16175,05002
EXPZ
OCTQ 02301,13760,05061,13402
EXQ
OCTQ 02457,17326,12466,02402
EXR
OCTQ 02650,02363,03177,03602
EXS
OCTQ 03053,03452,01052,12002
EXT
OCTQ 03272,02374,14532,15202
EXU
OCTQ 03526,00306,16373,12202
NAME: Logarithm

TEMPORARY STORAGE: COMMON, F2

OTHER SUBROUTINES USED: None

NUMBER OF WORDS: 53

EXECUTION TIME: 170 microseconds

ENTRY:
CALL LGT1 for log_{10}x
CALL LGEL for log_{e}x
CALL LGEL for log_{2}x

The number x, of which the logarithm is desired, should be in the accumulator; on exit, the appropriate logarithm replaces this number in the accumulator. If x ≤ 0, OV is set by the routine, and x is left in the accumulator; therefore OV should be cleared before entering.

EXIT:
Standard by JLH M3 with the appropriate log x in the accumulator.

METHOD:
To find the logarithm x, the routine normalizes x as x = f .4^n, 1/4 ≤ f < 1. An appropriate value of

$$\frac{16}{4^{\frac{1}{2} + k}} \quad (k = 0, 1, \ldots, 11)$$

is chosen from a stored table of values so that
METHOD (continued):

\[ f^1 = f \cdot \left[ \frac{16}{4 \frac{1}{2} + k} \right] \]

is such that \( \frac{8}{9} \leq f^1 < \frac{10}{9} \). A number

\[ a = \frac{f^{11}}{f^{11} + 2}, \quad (f^{11} = f^1 - 1) \]

is found, \( -\frac{1}{17} \leq a < \frac{1}{19} \), and following the series is computed for \( \log_e f^1 \):

\[ \log_e (f^1) = \log_e (1 + f^{11}) = \log_e (\frac{1 + a}{1 - a}) = 2a + \frac{2a^3}{3} + \frac{2a^5}{5} + \ldots + \frac{2a^{13}}{13} \]

Finally \( \log_e x \) is found from:

\[ \log_e x = \log_e (f^1) + n \log_e 4 - \log_e \left( \frac{16}{4 \frac{1}{2} + k} \right) \]

where \( \log_e \left( \frac{16}{4 \frac{1}{2} + k} \right) \) is found from a stored table.

\[ \log_{10} x = (\log_{10} 10)^{-1}(\log_e x); \quad \log_2 x = (\log_e 2)^{-1}(\log_e x). \]

RANGE:

Finds the logarithm of all \( x > 0 \). OV set if \( x \leq 0 \).

ACCURACY:

Exact to 12 decimal places, unless characteristic is zero in which case good to 11 decimal places. Small round-off error in the 13th or 12th decimal place as the case may be.
FIL
SFR 4, COMMON
CAM 1, 1
TRA 3, LGT1+3

LOG 10 X
M1=1 MEANS LOG 10 X

LGB1 SFR 4, COMMON
CAM 1
TRA 3, LGT1+3

LOG 2 X
M0=0 MEANS LOG 2 X

LGE1 SFR 4, COMMON
CSM 1, 1
TZN 3, LGT1+17
STR F2
SEX 3
CAE 0
ADE 2
SIA 0
SBE 2
MPY LGT1+15+MO

M1=-1 MEANS LOG E X
X LT EQUAL TO 0 MEANS ERROR
NORMALIZE X, X=F-4N

SUB 1.
STR F2
ADD 2.
VID F2
STR F2
MPY F2
STR F3
CAM 2,-6
CAD LGT1+43
FIL
MPY F3
ADD LGT1+50+M2
CJF 2, 0
MPY F2
SUB LGT1+27+M0
STR F2
CAD LGT1+50
MPY M3.

LGT10000
LGT10001
LGT10002
LGT10003
LGT10004
LGT10005
LGT10006
LGT10007
LGT10008
LGT10009
LGT10010
LGT10011
LGT10012
LGT10013
LGT10014
LGT10015
LGT10016
LGT10017
LGT10018
LGT10019
LGT10020
LGT10021
LGT10022
LGT10023
LGT10024
LGT10025
LGT10026
LGT10027
LGT10028
LGT10029
LGT10030
LGT10031
LGT10032
LGT10033
LGT10034
OCTQ  04730, 11661, 03542, 10377
OCTQ  05642, 16427, 04272, 03377
OCTQ  07070, 16161, 14343, 10577
OCTQ  02222, 04444, 11111, 02200
OCTQ  03146, 06314, 14631, 11400
OCTQ  05252, 12525, 05252, 12600
OCTQ  04000, 00000, 00000, 00001
OCTQ  02613, 11027, 17372, 03601
OCTQ  03362, 15730, 12446, 16400
OCTQ  02705, 05073, 02453, 10201

A6
A5
A4
A3
A2
A1
A0
LN 4
1 / EN 10
WILL BE 1 / EN 2

Date: 11/25/63
Section: 8.4-B-3-L7m1
Page: 5 of 5
Change:
NAME: Hyperbolic sine

OTHER SUBROUTINES USED: EXPL

TEMPORARY STORAGE: None

NUMBER OF WORDS: 21 words

EXECUTION TIME: $89.3 \mu\text{sec for } |x| \leq 1, 178.3 \mu\text{sec average for } 1 \leq |x| \leq 10.$

USE: Standard, CALL SINH1 ($x$ in the accumulator) and normal exit ($\sinh x$ in the accumulator). Fast registers are saved. OV is cleared if EXPL is called.

MATHEMATICAL METHOD: For $|x| \leq 1$, a Chebyshev polynomial organized by powers of $x$ is used. The coefficients are given in the reference below. For $|x| > 1$,

$$\sinh x = \frac{1}{2} (e^x - 1/e^x)$$

where $e^x$ is computed by EXPL.

ACCURACY: Using the identity $\sinh^2(2x) = 4\sinh^2 x(1 + \sinh^2 x)$, the maximum relative error is $2.2 \times 10^{-12}$ and the average error is $.30 \times 10^{-12}$.

SINH1 SFR 4, SINH1C  SAVE F4.
STR F4  SAVE X.
DAV -  FORM
SUB 1. ABS(X) - 1.
TP SINH1B JUMP IF X .L. -1 OR X .G. 1.
CAD F4  FORM
MPY F1  X**2.
SFR 5, SINH1C+1 SAVE F5.
CAM 4, SINH1D SET LOCATION OF POLYNOMIAL CONSTANTS.
STR F0  Y = X**2.
MPY 4,1, FORM A13 * Y.
CSM 5,5 SET LOOP COUNTER.
FLD
ADD 4,1, SINH1(X) IS
MPY F0  COMPUTED FROM
CJF 5  A CHEBYSHEV
ADD M4 POLYNOMIAL EXPANSION.
MPY F4  SINH1(X) = A1*X + A3*X**3+...+A13*X**13.
LFR 5, SINH1C+1 RESTORE F5.
SINH1A LFR 4, SINH1C RESTORE F4.
JLH M3  EXIT SINH1 SUBROUTINE.
SINH1B CAD F4  X.
CALL EXP1 SINH1(X) IS
STR F0  COMPUTED
VID -1. FROM THE
ADD F0  EXP1 SUBROUTINE.
MPY 10,3,2048 SINH1(X) = (1/2) * (E**X - E**(-X)).
TRA SINH1A JUMP TO EXIT.
SINH1C BSS 2 SINH1 TEMP STORAGE.
SINH1D OCTQ 05474, 00102, 04236, 16560 A13.
OCTQ 03271, 04232, 12511, 15164 A11.
OCTQ 05616, 17127, 11460, 16767 A9.
OCTQ 06400, 15001, 07005, 10372 A7.
OCTQ 04210, 10421, 01047, 02775 A5.
OCTQ 05252, 12525, 05252, 12377 A3.
OCTQ 02000, 00000, 00000, 00001 A1.

SINH1 33
SINH1 34
SINH1 35
SINH1 36
Nama: Square Root


Subrutin Lainnya: Tidak ada

Penyimpanan Sementara: F2 dan COMMON

Jumlah Kata: 9

Durasi: 150 mikrosekunder

Keakuratan: Kesalahan maksimum tidak lebih dari 1.5 dalam tempat angka signifikan terkecil (bit 44).

Kebutuhan Daerah: Exponent akumulator dapat berada dalam rentang -128 hingga +127 pada waktu masuk.

Masuk: Masuk dengan CALL SQRT dengan X dikunci di akumulator.

Keluar: Masuk dengan \(\sqrt{X}\) ke dalam akumulator jika \(X \geq 0\) atau dengan X dalam akumulator jika \(X < 0\).

Metode: Empat iterasi dari formula Newton \(X_{n+1} = \left(X_{n} + X/X_{n}\right)/2\) digunakan.

Penormalan awal \(X_0\) dibentuk dari bentuk normal \(X = \sqrt{2}^E\) sebagai berikut:

- Jika \(E\) ganjil, \(X_0 = (Y + 1)4^{(E-1)/2}\).
- Jika \(E\) genap, \(X_0 = 1/2(Y + 1)4^{E/2}\).

Iterasi dilakukan dengan satu zero exponent.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIL</td>
<td>SQUARE ROOT FIRST CARD</td>
</tr>
<tr>
<td>TZN</td>
<td>TEST FOR ZERO</td>
</tr>
<tr>
<td>SFR</td>
<td>SAVE F4</td>
</tr>
<tr>
<td>STR</td>
<td>ROUND</td>
</tr>
<tr>
<td>SEX</td>
<td>STORE EXPONENT</td>
</tr>
<tr>
<td>CAE</td>
<td>CLEAR EXPONENT</td>
</tr>
<tr>
<td>2,3,</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>ADD</td>
<td>ADD I</td>
</tr>
<tr>
<td>2,3,</td>
<td>DIVIDE BY FOUR</td>
</tr>
<tr>
<td>VID</td>
<td>SET COUNT</td>
</tr>
<tr>
<td>2,3,</td>
<td>STORE XN</td>
</tr>
<tr>
<td>ASC</td>
<td>X/XN</td>
</tr>
<tr>
<td>CAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>0,3,</td>
<td>DIVIDE BY FOUR</td>
</tr>
<tr>
<td>SBE</td>
<td>LOOP FOUR TIMES</td>
</tr>
<tr>
<td>CSM</td>
<td>HALVE EXPONENT</td>
</tr>
<tr>
<td>2,4,</td>
<td>TEST FOR EVEN EXPONENT</td>
</tr>
<tr>
<td>STR</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>0,3,</td>
<td>SET EXPONENT</td>
</tr>
<tr>
<td>CAT</td>
<td>RESTORE F4</td>
</tr>
<tr>
<td>0,3,</td>
<td>RESET OVERFLOW</td>
</tr>
<tr>
<td>ADE</td>
<td>EXIT</td>
</tr>
<tr>
<td>LFR</td>
<td>3, COMMON</td>
</tr>
<tr>
<td>TOR</td>
<td>3, SQUARE</td>
</tr>
<tr>
<td>J LH</td>
<td>3,</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

ILLIAC II Library Routine
J5-CCP4SC-40-UI-AL
August 19, 1965

IDENTIFICATION
CalComp Plotter Scale for FORTRAN and NICAP.

PURPOSE
CCP4SC (1) finds the minimum value in a specified subarray of a given one-dimensional array, and (2) computes a scaling factor for the array. It places these results into a two-element array which can be used with CCP6LM to graph the array and/or with CCP5AX to construct an axis.

RESTRICTIONS
None.

REGISTERS AND USER'S MEMORY CHANGED
M3 by the calling sequence, F0-F3, and accumulator.

TEMPORARY STORAGE
None.

LENGTH OF ROUTINE
53 words excluding other subroutines used.

OTHER SUBROUTINES USED
J5-DXDY-00-UI-AL

EXECUTION TIME
.1 seconds for 4,000 numbers.

ENTRY
The NICAP calling sequence is

\begin{verbatim}
CALL CCP4SC
DEQ X, S, N, K
DEQ T
\end{verbatim}

where X, S, N, K, and T are the addresses of the parameters defined below and not the parameters themselves.

Programmed by: Richard Lyon
Approved by: Joanne Watkins
IDENTIFICATION
CalComp Line for FORTRAN and NICAP.

PURPOSE
CCP6LN plots and connects on the CalComp Plotter a specified subset of the set of points

\[(X(I)-XMIN, Y(I)-YMIN), \quad I=1, 2, \ldots, N,\]
given the arrays X and Y, whose elements are X(1), X(2), \ldots, X(N) and Y(1), Y(2), \ldots, Y(N), respectively, and given XMIN, YMIN, DX and DY.

RESTRICTIONS
\[(y_{\text{max}}-y_{\text{min}})/DY < 29\]
where \(y_{\text{max}}\) is the maximum and \(y_{\text{min}}\) is the minimum of \(Y(1), Y(|K|+1), Y(2|K|+1), \ldots, Y(r|K|+1)\) where \(r\) is the largest integer such that \(r|K|+1 \leq N\).

REGISTERS AND USER'S MEMORY CHANGED
M3 by the calling sequence, accumulator and F0-F3.

TEMPORARY STORAGE
None.

LENGTH OF ROUTINE
About 81 words excluding other subroutines used.

OTHER SUBROUTINES USED
J5-CCPLPP-00-U1-AL

EXECUTION TIME
3.68 seconds for 2,000 data points.

ENTRY
The NICAP calling sequence is

\[
\begin{align*}
\text{CALL} & \quad \text{CCP6SC} \\
\text{DECQ} & \quad X,Y,N,K \\
\text{DECQ} & \quad TX,TY,
\end{align*}
\]

where X, Y, N, K, TX, and TY are addresses of the parameters defined below and not the parameters themselves.

Programmed by: Richard Lyon
Approved by: Joanne Watkins
TITLE: Gauss Quadrature

LENGTH: 194 words

TEMPORARY STORAGE: 18 words at a location specified by the calling program (see Method of Use).

OTHER SUBROUTINES USED: A subroutine to evaluate $f(x)$ supplied by calling program (see Method of Use).

TIMING: Ranges from 200 $\mu$sec for $n = 2$ to 875 $\mu$sec for $n = 15$. $(50n + 100)$ $\mu$sec is a good estimate of the timing.

PURPOSE: To evaluate numerically

$$\int_{p}^{q} f(x)dx$$

where $p, q$ are constants supplied by the calling program, and $f$ is a real-valued function which is evaluated by a subroutine also supplied by the calling program.

METHOD OF USE: The calling program must supply five items as follows:

1. $p$: The lower limit of integration must be in F2 upon entry to the subroutine.

2. $q$: The upper limit of integration must be in the accumulator upon entry to the subroutine.

3. $n$: The number of points at which the function is to be evaluated must be in the first quarter-word of the word following the CALL instruction; $n$ must be an integer in the range 2, 3, ..., 16.
METHOD OF USE (cont'd): 4. f: Auxiliary subroutine to evaluate \( f(x) \) at arguments supplied by this subroutine. Upon calling the auxiliary subroutine, the argument \( x \) is placed in the accumulator, and \( f(x) \) is to be returned to the accumulator. The auxiliary subroutine must begin in the left-most quarter-word. The address of this location is to be placed in the second quarter-word of the word following the CALL instruction. The link to the auxiliary subroutine is M3. The control word following CALL has the form:

| n | address of \( f \) | not used | not used |

5. A block of 18 consecutive words for temporary storage. The address of the first word of the block must be put in M2 before entering the subroutine.

EXIT: Upon exit from this subroutine, the approximation to the integral of the function is returned to the accumulator. [The accumulator will contain

\[
\frac{q-p}{2} \sum_{i=1}^{n} a_i f\left( x_i \frac{q-p}{2} + \frac{q+p}{2} \right).
\]

Control is returned to the word following the one which contains the parameters.

EXAMPLE: A portion of program which might be used to call this subroutine is as follows:

```
CAM 2,400 400 is the location of block for temporary storage
CAD 6.
STR F2 p = 6 (lower limit of integration)
CAD 9. q = 9 (upper limit of integration)
CALL GQU1
CONTROL DECQ 10,250, n = 10, the auxiliary subroutine f is located beginning at 250
here
```
MATHEMATICAL DESCRIPTION: The Gaussian quadrature formula for evaluating an integral with arbitrary limits \((p, q)\) is given by

\[
\int_{p}^{q} f(x) \, dx = \frac{q - p}{2} \sum_{i=1}^{n} a_i f\left( x_i \frac{q - p}{2} + \frac{p + q}{2} \right) + R_n(f)
\]

where \(x_i\) is the \(i\)th root of \(P_n(x)\) and

\[
a_i = \frac{1}{P_n'(x_i)} \int_{-1}^{1} \frac{P_n(x)}{x - x_i} \, dx
\]

\[x_i = -x_{n-i+1}, \quad a_i = a_{n-i+1}\]

If \(f_{2n-1}(x)\) is an arbitrary polynomial of degree at most \(2n - 1\), then \(R_n(f_{2n-1}) = 0\).

If \(f(x)\) has a continuous derivative of order \(2n\) in the interval \((p, q)\), then

\[
R_n = \frac{f^{(2n)}(\xi)}{(2n)! K_n^2}
\]

where \(\xi\) is a point in the interval \((p, q)\) and

\[
K_n = \frac{(2n)_n (2n+1)^{1/2}}{(q - p)^{n+1/2}}.
\]

\(R_n\) is the error term. The zeros of the Legendre polynomials \(P_n(x)\), \(n = 2, 3, \ldots, 16\), and the corresponding weight coefficients \(a_i\) were taken from the 16 place tables from Tables of Functions and of Zeros of Functions, Applied Mathematics Series 37, prepared by the U.S. Department of Commerce, National Bureau of Standards.
FIL
GQU1
STR  F3
ADM  2,14
ATN  2,10
SFR  4
ATN  2,10
SFR  5
ATN  2,10
SFR  6
ATN  2,10
SFR  7
LFR  5,M3
CAD  F3
SUB  F2
MPY  10,3,2048
STR  M2-6
ADD  F2
STR  M2-5
CRM  4,1
JNM  4,GQU1+38
CRM  4,12
CSB  M4.
MPY  10,3,2048
SIA  8
CAM  1
CAD  M4.
SUB  2.
TZ   GQU1+11
CJU  1,2,GQU1+9
CAD  M1.
MPY  M1+1.
ADD  GQU1+59.
SIA  0
ADD  M1+64.
SIA  12

PICK UP PARAMETERS N AND F
(Q-P)/2
(Q+P)/2
JUMP IF N IS ODD
LOCATE XI IN TABLE
LOCATE AI IN TABLE
FORM ARGUMENT T+R

EVALUATE F(T+R):

EVALUATE F(-T+R):

FORM: SUMMATION
MPY  M2-6
LFR  4,M2-4
ATN  2,19
LFR  5
ATN  2,19
LFR  6
ATN  2,19
LFR  7
ADM  2,-18
JLH  3,2,1
CAD  M2-14
TRA  1,GQU1+33
CRM  4,12
CAM  9
CSB  M4-1.
MPY  10,3,2048
SIA  8
CAM  1,1
CAD  M4-
SUB  3.
TZ   2,GQU1+44
ADM  1,1
SUB  2.
TZ   2,GQU1+44
CJU  1,GQU1+43
CAD  M1.
MPY  M1+1.
ADD  GQU1+122.
SIA  12
SUB  M1+GQU1+122.
ADD  GQU1+59.
SIA  0
SFR  4,M2-18
SFR  5,M2-17
SFR  6,M2-16

EXIT

LOCATE FIRST A

LOCATE FIRST X
<table>
<thead>
<tr>
<th>SFR</th>
<th>7,M2-15</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAD</td>
<td>M2-5</td>
</tr>
<tr>
<td>JSB</td>
<td>3,M5</td>
</tr>
<tr>
<td>FIL</td>
<td></td>
</tr>
<tr>
<td>LFR</td>
<td>4,M2-18</td>
</tr>
<tr>
<td>LFR</td>
<td>5,M2-17</td>
</tr>
<tr>
<td>LFR</td>
<td>6,M2-16</td>
</tr>
<tr>
<td>LFR</td>
<td>7,M2-15</td>
</tr>
<tr>
<td>ATN</td>
<td>12,1</td>
</tr>
<tr>
<td>MPY</td>
<td>0</td>
</tr>
<tr>
<td>CAM</td>
<td>6,M2-14</td>
</tr>
<tr>
<td>STR</td>
<td>6,1</td>
</tr>
<tr>
<td>TRA</td>
<td>2,GQU1+15</td>
</tr>
</tbody>
</table>

**EVALUATE F((Q+P)/2)**

<table>
<thead>
<tr>
<th>OCCQ</th>
<th>04474, 15164, 05440, 06400, 06144, 13757, 04176, 16200, 02560</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCCQ</td>
<td>10776, 14735, 03600, 06707, 03333, 14754, 07400, 04235, 11077</td>
</tr>
<tr>
<td>OCCQ</td>
<td>15463, 06600, 07177, 13316, 00246, 07600, 07505, 10455, 00154</td>
</tr>
<tr>
<td>OCCQ</td>
<td>16777, 05224, 05011, 06253, 11000, 07353, 06244, 16401, 04200</td>
</tr>
<tr>
<td>OCCQ</td>
<td>03176, 05357, 10432, 00000, 05735, 04771, 13707, 04000, 07457</td>
</tr>
<tr>
<td>OCCQ</td>
<td>10570, 16370, 14000, 05675, 06224, 17170, 16577, 04150, 11225</td>
</tr>
<tr>
<td>OCCQ</td>
<td>10145, 00600, 06277, 02253, 02223, 16400, 07535, 05434, 13025</td>
</tr>
<tr>
<td>OCCQ</td>
<td>11600, 02460, 02213, 07110, 12600, 04720, 05722, 00357, 17600</td>
</tr>
<tr>
<td>OCCQ</td>
<td>06540, 06104, 17032, 02000, 07575, 11626, 16576, 04400, 04607</td>
</tr>
<tr>
<td>OCCQ</td>
<td>02407, 10443, 14377, 03357, 03000, 04401, 06400, 05336, 15622</td>
</tr>
<tr>
<td>OCCQ</td>
<td>02433, 14400, 06727, 04625, 16471, 06200, 07625, 01740, 06125</td>
</tr>
<tr>
<td>OCCQ</td>
<td>16200, 02120, 00617, 07753, 04000, 04116, 03367, 14233, 01400</td>
</tr>
<tr>
<td>OCCQ</td>
<td>05656, 13173, 15103, 02600, 07061, 06421, 12740, 11200, 07646</td>
</tr>
<tr>
<td>OCCQ</td>
<td>15142, 17657, 15400, 04003, 15137, 10765, 06377, 02742, 12151</td>
</tr>
<tr>
<td>OCCQ</td>
<td>00250, 13000, 04545, 12360, 05600, 17000, 06121, 10256, 16661</td>
</tr>
<tr>
<td>OCCQ</td>
<td>01400, 07167, 04164, 17767, 01000, 07664, 07435, 10710, 05600</td>
</tr>
<tr>
<td>OCCQ</td>
<td>07277, 15207, 11036, 05377, 03455, 00327, 10214, 07200, 05107</td>
</tr>
<tr>
<td>OCCQ</td>
<td>01003, 03540, 03400, 06323, 04161, 10470, 17000, 07265, 07565</td>
</tr>
<tr>
<td>OCCQ</td>
<td>01341, 11400, 07677, 03327, 06122, 14600, 03352, 05751, 13336</td>
</tr>
<tr>
<td>OCCQ</td>
<td>07577, 02433, 01262, 04422, 15200, 04076, 07253, 05235, 16000</td>
</tr>
<tr>
<td>OCCQ</td>
<td>05377, 02333, 00474, 01600, 06474, 03356, 04333, 13200, 07332</td>
</tr>
<tr>
<td>OCTQ</td>
<td>01177, 02200, 16023, 00265, 15177, 03737, 07071, 01172, 02576</td>
</tr>
<tr>
<td>OCTQ</td>
<td>06037, 17256, 10034, 11577, 05657, 14307, 07701, 10177, 05323</td>
</tr>
<tr>
<td>OCTQ</td>
<td>07273, 13264, 16777, 04622, 17541, 13624, 07577, 03771, 16571</td>
</tr>
<tr>
<td>OCTQ</td>
<td>05655, 03577, 03027, 01167, 03634, 14377, 07757, 15432, 02602</td>
</tr>
<tr>
<td>OCTQ</td>
<td>04776, 03363, 07325, 12267, 14176</td>
</tr>
</tbody>
</table>

Date: 11/19/63
Section: 8.4 FL-9040
Page: 9 of 9
Change:
NAME: Runge-Kutta-Gill

PURPOSE: To solve a system of N simultaneous, first-order, ordinary differential equations.

OTHER SUBROUTINES USED: An auxiliary subroutine provided by the programmer.

TEMPORARY STORAGE: Three consecutive words beginning at a location given in M2.

NUMBER OF WORDS: 25

FAST REGISTERS CHANGED: F2

EXECUTION TIME: $50 + 290N + \frac{4}{N}$ (auxiliary subroutine time in microseconds) where N is the number of equations to be solved.

USE: Standard by CALL RKGl with: the address of the first of three words of temporary storage in M2; the parameters

\[
\begin{align*}
A & \quad M & \quad N \\
\end{align*}
\]

in the word immediately following the CALL, where A is the address of the auxiliary subroutine (which must begin in the first quarter of a word), N is the number of equations to be solved, and M is the address of the first word of a block of 3N words to be used by RKGl.

(a) $y_0, y_1, \ldots, y_{N-1}$ are stored in locations $M, M+1, \ldots, M+N-1$ respectively. The initial conditions are stored here by the user. The auxiliary subroutine uses $y_0, y_1, \ldots, y_{N-1}$ (but does not alter them) to compute the $k_1$'s. The solutions are found here upon return from RKGl.
USE (Continued):

(b) \( k_0, k_1, \ldots, k_{N-1} \) are stored in locations M+N, M+N+1, \ldots, M+2N-1 respectively. The \( k_1 = hf \) are computed and placed here by the auxiliary subroutine.

(c) \( q_0, q_1, \ldots, q_{N-1} \) are stored in locations M+2N, M+2N+1, \ldots, M+3N-1 respectively. (Note: This block of N words must be cleared to zero by the user prior to his first entry into RKGL.)

If the independent variable \( x \) occurs in the functions \( f_i \) or if it is required during an integration as an index, then it must be obtained by integrating the equation \( x' = 1 \). The independent variable is then treated as an additional dependent variable, for which the auxiliary subroutine must provide the quantity \( hx' = h \). However, this latter quantity should be planted at the beginning of the integration in the appropriate location and left there so that the auxiliary subroutine is relieved of the task. If \( x \) does not appear in any of the \( f_i \)'s but is merely wanted for indication purposes, it is quicker to use a simple counter in the main routine.

Backward integration is achieved by making \( h \) negative.

METHOD:

Given the set of \( N \) differential equations

\[
y'_i = f_i(y_0, y_1, y_2, \ldots, y_{N-1}) \quad (i = 0, 1, 2, \ldots, N - 1)
\]

the process used in the integration is defined by the following equations:

\[
k_{ij} = hf_i(y_{ij}, y_{lj}, \ldots, y_{(N-1)j})
\]

\[
r_{i,j+1} = B_j(k_{ij} - A_j q_{ij})
\]

\[
y_{i,j+1} = y_{ij} + r_{i,j+1}
\]
\[ q_{i,j+1} = q_{i,j} + 3r_{i,j+1} - C_j k_{i,j} \]

with the following table of values:

<table>
<thead>
<tr>
<th>j</th>
<th>A_j</th>
<th>B_j</th>
<th>C_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>(1/2)</td>
<td>(1/2)</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>(1 - \sqrt{1/2})</td>
<td>(1 - \sqrt{1/2})</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>(1 + \sqrt{1/2})</td>
<td>(1 + \sqrt{1/2})</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>(1/6)</td>
<td>(1/2)</td>
</tr>
</tbody>
</table>

where the subscript \(i\) indicates the variable, the subscript \(j\) indicates the four parts of the integration step size. The process is sometimes known as the Gill-Kutta method.

The \(k_{i,j}\)'s are evaluated by a closed auxiliary subroutine which must be provided by the user. During each pass through RKGL four entries are made into the auxiliary routine to obtain the \(k_{i,j}\)'s, \(k_{2,j}\)'s, \(k_{3,j}\)'s and \(k_{4,j}\)'s.

RKGL does the arithmetic indicated in the \(r_{i,j}\), \(y_{i,j}\) and \(q_{i,j}\) equations above.

This is a fourth-order process, hence the truncation error in one step is of the order of \(h^5\). An approximation to the error is obtained from the expression

\[ \frac{1}{15}(y_h - y_{h/2}) \]

where \(y_h\) is the value of \(y\) obtained from using an interval of length \(h\) and \(y_{h/2}\) is the value of \(y\) obtained from using an interval of length \(h/2\). The rapid accumulation of round-off errors is suppressed by retaining the quantities \(q_i\) between integration steps.
RKG1
ATN  2,1,
SFR  7
ATN  3,1,
LFR  7
ATN  2,1,
SFR  6
ATN  2,1,
SFR  4
cam  10,RKG1+17
CAM  8,M14+M14
CSM  9,4
JSB  3,M12
FIL
CSM  11,M14
CSB  M8+M13
MPY  10,1,
ADD  M13+M14
MPY  10,0,
STR  2,3,
ASC  13,0,
CAD  2,3,
MPY  3
STR  2,3,
CSB  M13+M14
ADN  9,1
CAM  15
MPY  10,1,
JNM  15,RKG1+11
MPY  3
ADD  2,3,
ATN  8,0,
ASC  13,1,
F7=PARAMETERS FOR RUNGE-KUTTA
M10 = ADDRESS OF FIRST CONSTANT
M8 = 2N
M9 = -4, STEP COUNTER
ENTER AUXILIARY
M11 = -N
ACCU = -QIJ
ACCU = -AJ*QIJ
ACCU = KIJ - AJ*QIJ
ACCU = BJ(KIJ - AJ*QIJ) = RIJ
F2 = RIJ
YI, J+1 = YI + RIJ
ACCU = FIJ
ACCU = 3RIJ
F2 = 3RIJ
ACCU = -KIJ
M15 = 0 ON 4TH STEP, ELSE-VE.
ACCU = BJ*KIJ, INCREMENT M10
JUMP IF NOT 4TH STEP
ACCU = (-1/6)K14 - (1/2)K14
ACCU = 3RIJ-BJ*KIJ
QI, J+1 = QIJ + 3RIJ - BJ*KIJ
<table>
<thead>
<tr>
<th>CJZ</th>
<th>11, 1, RKG1+13</th>
<th>COUNT AND JUMP IF I = N-1</th>
<th>RKG1 032</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBM</td>
<td>10, 2</td>
<td>RESET M10</td>
<td>RKG1 033</td>
</tr>
<tr>
<td>TRA</td>
<td>2, RKG1+5</td>
<td>JUMP TO CALCULATE Y1+1, J+1, Q1+1, J+1</td>
<td>RKG1 034</td>
</tr>
<tr>
<td>SBM</td>
<td>13, M14</td>
<td>RESET M13 = A</td>
<td>RKG1 035</td>
</tr>
<tr>
<td>CJU</td>
<td>9, 3, RKG1+3</td>
<td>COUNT AND JUMP IF J NOT 4</td>
<td>RKG1 036</td>
</tr>
<tr>
<td>SBM</td>
<td>2, 3</td>
<td></td>
<td>RKG1 037</td>
</tr>
<tr>
<td>ATN</td>
<td>2, 1</td>
<td></td>
<td>RKG1 038</td>
</tr>
<tr>
<td>LFR</td>
<td>7</td>
<td></td>
<td>RKG1 039</td>
</tr>
<tr>
<td>ATN</td>
<td>2, 1</td>
<td></td>
<td>RKG1 040</td>
</tr>
<tr>
<td>LFR</td>
<td>6</td>
<td></td>
<td>RKG1 041</td>
</tr>
<tr>
<td>ATN</td>
<td>2, 1</td>
<td></td>
<td>RKG1 042</td>
</tr>
<tr>
<td>LFR</td>
<td>4</td>
<td></td>
<td>RKG1 043</td>
</tr>
<tr>
<td>SBM</td>
<td>2, 3</td>
<td></td>
<td>RKG1 044</td>
</tr>
<tr>
<td>JLH</td>
<td>3, 0</td>
<td></td>
<td>RKG1 045</td>
</tr>
<tr>
<td>FIL</td>
<td></td>
<td></td>
<td>RKG1 046</td>
</tr>
<tr>
<td>OCTQ</td>
<td>4000, 1, 4000</td>
<td></td>
<td>RKG1 047</td>
</tr>
<tr>
<td>OCTQ</td>
<td>2000, 1, 1453</td>
<td>16606, 6300, 6201</td>
<td>RKG1 048</td>
</tr>
<tr>
<td>OCTQ</td>
<td>2000, 1</td>
<td></td>
<td>RKG1 049</td>
</tr>
<tr>
<td>OCTQ</td>
<td>3324, 1171, 11477, 11601, 4000, 1</td>
<td>RKG1 050</td>
<td></td>
</tr>
<tr>
<td>OCTQ</td>
<td>5252, 12525, 5252, 12577</td>
<td></td>
<td>RKG1 051</td>
</tr>
</tbody>
</table>
NAME: Divided differences

OTHER SUBROUTINES USED: An auxiliary routine for evaluating the function f(x).

TEMPORARY STORAGE: A block of \((k + 3)\) consecutive words, beginning at the location specified by the content of M2 at entry time.

NUMBER OF WORDS: 15 words \((14 \times 4\) quarter words if the routine begins in quarter word 1)

FAST REGISTERS CHANGED: F2

PARAMETERS: Link in M3.
Address of first location of temporary storage block in M2.

Three parameters which have to be written in the word following the one with the CALL instruction:

\[
\begin{array}{c}
  f \quad \text{address of auxiliary} \\
  A \quad \text{address of first abscissa} \\
  k \quad \text{kth divided difference}
\end{array}
\]

EXECUTION TIME: Dependent on the parameter \(k\) and the duration \(T_A\) of the auxiliary routine (which is entered \(k + 1\) times).
Approximately, the total duration \(T\) is given by

\[
T \approx k^2 \cdot 25\ \mu\text{sec} + (k + 1)T_A
\]

Programmed by: Jürg Nievergelt
Approved by: 

Date: 7/20/64
Section: 8.4-EL-DVDF1
Page: 1 of 4
Change:
USE:

The user must provide:

1) $k+1$ abscissas $X_0$, $X_1$, ..., $X_k$
in locations $A$, $A+1$, ..., $A+k$

2) An auxiliary routine which takes the argument $X_j$
either from the accumulator or from F1 and leaves
\( f(X_j) \) in the accumulator, has to be entered in
quarter word 0 and linked in M3.

3) A block of $k+3$ consecutive words beginning at the
location specified by the value of M2 at entry time.

The subroutine computes a divided difference table and
stores:

- kth divided difference \( f[X_0, \ldots, x_k] \) in location
  \( (M2) + 2 \)
- (k-1)st divided difference \( f[x_1', \ldots, x_k'] \) in location
  \( (M2) + 3 \)
- (k-2)nd divided difference \( f[x_2', \ldots, x_k'] \) in location
  \( (M2) + 4 \)

\[ \vdots \]
- 1st divided difference \( f[X_{k-1}', X_k'] \) in location
  \( (M2) + k + 1 \)

the value of the function \( f(X_k) \) in location \( (M2) + k + 2 \)

where \( (M2) \) is the content of M2 at entry time. The kth
divided difference \( f[X_0', x_1', \ldots, x_k'] \) is also left in
the accumulator.

REMARK:

This routine can be used recursively, i.e., the auxiliary
routine \( f \) may again contain a CALL DIVF.
DIVDF1 ATN 2,1,  DIVIDED DIFFERENCES FIRST CARD
SFR 5       SAVE F5
ATN 3,1,   INCREASE LINK BY 1
LFR 5      READ PARAMETERS INTO F5
ATN 2,1,    SAVE F4
SFR 4       M0 = X0
ATN M5
ATN M6
ATN M6
CSM 7,1     M7 = -(K+1)
DIVF1 CAD 0,1,  ABSCISSA
ATN M4       AUXILIARY
JSB 3,0,0    FILL UP S0,S1,...,SK
FIL
STR 2,1,     M3 = -K
CJU 7,DIVF1
ATN M6
ATN M6
ATN M6
ATN M6
CSM 3
DIVF2 CAM 0    MI COUNTS UP TO K FOR INNER LOOP
ATN M3
ATN M6
ATN M6
CAM 1,1
DIVF3 ATN M5
CSB M0    CURRENT ABSCISSA
ATN M5
ATN M5
ADD 1,1,    NEXT ABSCISSA
STR F2     STORE DIFFERENCE OF ABSCISSAS IN F2
ATN M2
ATN M2
CSB 0,1,    CURRENT DIVIDED DIFFERENCE
ATN M2
NAME: Lagrange six-point interpolation for equal intervals.

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: COMMON to COMMON + 7.
On exit, COMMON to COMMON + 5 contain 120 times the six Lagrange coefficients A_{-2}, A_{-1}, A_0, A_2 and A_3 respectively.
1/120 is available at location LAG6 + 26.
M1 is incremented by the number of control words used.

NUMBER OF WORDS: 27

FAST REGISTERS CHANGED: F2, F3 and F4 (except MO and M2)

TIME: 250 + 125N microseconds where N is the number of control words.

ENTRY: Standard by CALL with:

x in Acc where x is the interpolation point scaled as if the tabulated values of f(x) are at x = 0, x = 1, x = 2, ..., and with the address of the first control word in M1.

The format of the control words is:

```
B S R C
```

where B is the base of the table, S is the spacing between the entries: that is, Y(0) is in B, Y(1) in B + S ... and Y(N - 1) is in B + (N - 1)S, R is the storage location where the result is to be placed, and if C ≠ 0, there is another control word specifying another table in the next higher addressed location.
EXAMPLE:

Suppose locations 100 to 199 contain

\[ \sin \left( \frac{n \pi}{400} \right) \quad n = 0, 1, \ldots, 99 \]

and locations 300 to 399 contain

\[ \tan \left( \frac{n \pi}{400} \right) \quad n = 0, 1, \ldots, 99 \]

To find \( \sin \alpha \) and \( \tan \alpha \), and to store in locations \( B \)
and \( B + 1 \) where \( \alpha \) is in the accumulator, the program
below can be used.

```
MPY   400
CAM   1, A
CALL   LA06
-------
FIL
A   DECQ 100, 1, B, 1
DECQ 200, 1, B + 1, 0
```

EXIT:

Standard by JLH M3. The Interpolated value of the last
table entry is in the accumulator at exit as well as
in the specified storage location.

RANGE:

Table entries \( f \) should satisfy \( |f| < \frac{1}{120} \times 4^{63} \) and \( x \) should
satisfy \( 2 \leq x < n - 3 \) where \( n \) is the number of table
entries.

ACCURACY:

Error arises from four sources. For a discussion see
"Tables of Lagrangian Interpolation Coefficients;"
The errors are:

1. Round-off in this subroutine \( < 11 \times 2^{-24}|f| \) where
   \( |f| \) is the maximum absolute table entry used.

2. Truncation in six-point Lagrange formula.
3. Error due to errors in table entries. This is bounded by $89/64 \max |\epsilon_i|$ where $\epsilon_i$ are the errors in the six entries used.

4. Error due to error $\varepsilon$ in interpolant $x$.

USE:

This subroutine can be used for interpolating simultaneously in several tables for different functions of the same argument $x$. If the functions are tabulated at points $x_0$, $x_0 + h$, $x_0 + 2h$, ..., $x_0 + nh$, $x$ must be translated and scaled before entry by subtracting $x_0$ and dividing by $h$. The user must prepare a list of control words in memory, one for each table, in ascending addresses. The last one should have a zero fourth quarter; all others should not.

METHOD:

120 times the six-point Lagrange coefficients

\[
A_{-2} = -(p - 3)(p - 2)(p - 1)p(p + 1)/120 \\
A_{-1} = +5(p - 3)(p - 2)(p - 1)p(p + 2)/120 \\
A_0 = -10(p - 3)(p - 2)(p - 1)(p + 1)(p + 2)/120 \\
A_1 = +10(p - 3)(p - 2)(p + 1)(p + 2)/120 \\
A_2 = -5(p - 3)(p - 1)p(p + 1)(p + 2)/120 
\]

and

\[
A_3 = +(p - 2)(p - 1)p(p + 1)(p + 2)/120 
\]

are calculated using 16 multiplies and placed in locations COMMON to COMMON + 5 where $p$ is the fraction part of $x$. Then for each table $j$,

\[
T_{j1} = \text{contents of location } (B_j + S_j(q - i)) 
\]

is found, and $\sum_{i=-2}^{+3} T_{j1}A_i$ is stored in location $R_j$ where $q$ is the integer part of $x$. 

STR     F3             (LAG6+13, 2)
MPY     M5
STR     5,1,             A(I), I=-2,-1,...,+2
CAD     -3-M6.
ADD     F2
MPY     F3
CJU     6,2;LAG6+13
STR     COMMON+5         A(3)
CAM     7,M4             M7=Q
ATN     1,1,             GET NEXT CONTROL WORD
LFR     6
CAD     M9.
MPY     -2+M7.
SIA     4
ADM     8,M4             (Q-2)*S+8
CAM     6,-6
CAD     15,3,
STR     F3
CAM     5,COMMON
CAD     5,1,             (LAG6+21,1)
MPY     M8
ADM     8,M9             A-3 T-3+...+A2T2
ASC     F3
CJU     6,1;LAG6+21
ADD     F3
MPY     LAG6+26          * 1/120
STR     M10             STORE RESULT
JUM     11,0;LAG6+17     TEST FOR MORE TABLES
LFR     5,COMMON+6
LFR     6,COMMON+7
JLH     M3
FIL
OCTQ    4210,10421,1042,2375  1/120
UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
ILLIAC II LIBRARY PROGRAM

EL-LGUN-00-UI-AL

NAME: Lagrange Interpolation for Unequal Intervals

TEMPORARY STORAGE: COMMON to COMMON + 1.

F2, F3 and F4 (except for M0 and M2).

M1 is incremented by the number of control words used.

OTHER SUBROUTINES USED: None

NUMBER OF WORDS: 28

EXECUTION TIME: $24N^2 + 28NM$ microsecs where $N$ is number of points used and $M$ is the number of functions interpolated.

USE: Standard by CALL LGUN with:

x in accumulator.

Control word list address in M1, where the First Control Word contains:

```
B x S x N T
```

where the values $x_0$ to $x_{N-1}$ of the independent variables x are in locations $B_x$, $B_x + S_x$, ..., $B_x + (N - 1)S_x$ respectively where $N$ is the number of points used and $T$ is the first word of a block of $N$ temporary storage locations in which the Lagrange coefficients $A_0$ to $A_{N-1}$ will be placed. The second and subsequent control words have the format

```
B S R C
```

where the functional values

$$T_0 = f(x_0), T_1 = f(x_1), \ldots, T_{N-1} = f(x_{N-1})$$
USE (continued): are in locations B, B + S, ..., B + S(N - 1) respectively. The result of interpolation in this table to get f(x), i.e., the number

\[ \sum_{i=0}^{N-1} A_i T_i \]

is stored in location R. If c \( \neq O \), there is another control word for another tabulated function in the next higher addressed location.

EXIT:

Standard by JIH M3 with the last interpolated value in the accumulator as well as in store, and Locations T, T + 1, ..., T + N - 1 contain the Lagrange coefficients \( A_0, A_1, ..., A_{N-1} \) respectively.

METHOD:
The N Lagrange coefficients \( A_0, A_1, ..., A_{N-1} \) are calculated, where

\[ A_j = \prod_{\substack{i=0 \backslash i \neq j \backslash i=0}}^{N-1} \frac{(x - x_i)}{(x_j - x_i)} \]

These appear in locations T + j.

This takes \( 3(N - 2) + N(N - 2) \) multiplications and N divisions.

Then for each table, \( \sum_{i=0}^{N-1} T_i A_i \) is evaluated.

ACCURACY:

Error can arise in five ways.

1. Accumulated round-off in this subroutine

\[ \leq \left( 4N + \sum_{i=0}^{N-1} |A_i| \right) 2^{-45} |f| \]

where \( |f| \) is the maximum absolute table entry.

3. Error in the interpolant.

4. Error in the table entries.

5. Error in the \( x_i \). If the \( x_i \) are close to each other, this error can be very large since it depends on terms including

\[
\frac{1}{x_i - x_j}
\]

RANGE:

The \( A_i \) will be in range provided that

\[
4^{-64} < |d|^{N-1} < |D|^{N-1} < 4^{-63}
\]

where \( d \) is minimum distance between the \( x_i \) and \( D \) is the maximum distance between the \( x_i \) or \( x_i \) and \( x \).

For a "rule of thumb" that relaxes this strict limit, the average distance between the \( x_i \)'s should be in the range

\[
\left(4^{-64}\right) \frac{1}{N-1}, \quad \left(\frac{4^{-63}}{(N-1)!}\right)^{\frac{1}{N-1}}
\]

The intermediate scalar products \( \sum A_i n \) cannot overflow if \( \sum |T_i| \ |D|^{N-1} \ < 4^{-63} \), although it is normally sufficient that the result be in range. If the strict bounds are not satisfied, overflow should be checked.

EXAMPLE:

Suppose that a monotone increasing function \( f(x) \) is tabulated for points \( x_0, x_1, \ldots, x_{99} \) in locations 100-199, that \( x_0, x_1, \ldots, x_{99} \) are in locations 200-299.
and that another function \( g(x) \) is tabulated in locations 300-399. Given \( \bar{f} \) it is desired to find \( \bar{x} \) and \( g(\bar{x}) \) where \( x \) is such that

\[
\bar{f} = f(\bar{x})
\]

Suppose that \( N \) is such that

\[
f(x_N) \leq \bar{f} \leq f(x_{N+1})
\]

and a six-point interpolation is employed the entry below can be used.

\[
\begin{array}{ll}
\text{CAM} & 1, A \\
\text{CALL} & \text{Lgun} \\
\text{DECQ} & N + 98, 1, 7, \text{COMMON} + 2 \\
\text{DECQ} & N + 198, 1, X, 1 \\
\text{DECQ} & N + 298, 1, G, 0
\end{array}
\]

COMMON is assumed to denote a block of eight storage locations, \( X \) contains \( \bar{x} \) and \( G \) contains \( g(\bar{x}) \) after execution. In this case we must have \( 2 \leq N \leq 97 \), so that the points used are inside the table.
LGUN
FIL:
SFR 5, COMMON
SFR 6, COMMON+1
STR F2
ATN 1, 1
LFR 6
SUB M8
CAM 4, 2-M10
CAM 5, 1+M11
CAM 6, M8
STR 5, 1
ADM 6, M9
CAD F2
SUB M6
MPY F0
CJU 4, LGUN+4
STR M5
ADM 6, M9
CAM 4, 2-M10
CAD F2
SUB M6
STR F3
SBM 5, 1
MPY M5
STR M5
CAD F2
SBM 6, M9
SUB M6
MPY F3
CJU 4, LGUN+8
STR M11
CAM 7, M8
CSM 6, M10
CAD M7
STR F2
F2=X(J)

FIRST CONTROL WORD
X=X0
M4=-N+2
M5=T+1
M6=B
(X-X(0))...(X-X(N-2))
(X-X(N-1))...(X-X(1))

LGUN0000
LGUN0001
LGUN0002
LGUN0003
LGUN0004
LGUN0005
LGUN0006
LGUN0007
LGUN0008
LGUN0009
LGUN0010
LGUN0011
LGUN0012
LGUN0013
LGUN0014
LGUN0015
LGUN0016
LGUN0017
LGUN0018
LGUN0019
LGUN0020
LGUN0021
LGUN0022
LGUN0023
LGUN0024
LGUN0025
LGUN0026
LGUN0027
LGUN0028
LGUN0029
LGUN0030
LGUN0031
LGUN0032
LGUN0033
LGUN0034
CAM 5, M8
SUB M5
SFN M10
CAM 4
JDC 4, 3, LGUN+17
CAD F2
ADM 5, M9
SUB M5
CJU 4, 3, LGUN+17
CAD F2
ADM 5, M9
SUB M5
JDC 5, LGUN+18
MPY F10
STR F10
CJU 4, LGUN+16
CAD F10
VID M11
STR 11, 1,
ADM 7, M9
CJU 6, 1, LGUN+12
SBM 11, M10
ATN 1, 1,
LFR 5
CAM 9, 2, 1-M10
CAD 11, 1,
MPY M4
STC F2
STR F0
ADM 4, M5
CAD M4
MPY 11, 1,
ADD F0
ASC F2
CJU 9, 3, LGUN+22

M5 = B 
X(J) - X(I)
M4 = -N 
IS X(J) - X(0) = 0
X(1) - X(O)
X(j) - X(I)
NEXT CONTROL WORD
M9 = (N-1)
A(O) T(O)
SUM A(I)T(I)
ADD F2
STR M6
JUM 7,1, LGUN+20
LFR 5, COMMON
LFR 6, COMMON+1
JLH 3,

TEST FOR MORE CONTROL WORDS
RELOAD F5
RELOAD F6
NAME: Simultaneous Linear Equations

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: Common to Common + 4 + N where N is the rank of the matrix A.

NUMBER OF WORDS: 73

FAST REGISTERS CHANGED: F2 and F3

EXECUTION TIME: Approximately $10N^2(N + 2M)$ microseconds

ENTRY: Standard by CALL SLQL with:

\[ P \begin{bmatrix} N & M & A & B \end{bmatrix} \]

N = dimension of matrix A.

M = number of solutions to computer, i.e., number of columns of matrix B.

A = address of first word of matrix A where A is stored consecutively by rows.

B = address of first word of matrix B where B is stored consecutively by rows.

For example, if we have

\[ P \text{ DECQ } 17,4,1000,1800 \]

and we have

Programmed by: J. Presti
Approved by: [Signature]

Date: 7/20/64
Section: 8.4-F4-SLQL
Page: 1 of 7
Change:
ENTRY (Continued):

CALL SLQ

then SLQ finds the four sets of solutions to the set of 17 simultaneous equations represented in core by the matrix beginning at 1000 and where the constant matrix begins at 1800.

EXIT:

Standard by JLH M3. The value of the determinant of the matrix is in the accumulator at exit.

METHOD:

The solutions for the matrix problem

\[ AX = B \]

are obtained via Gauss-Jordan elimination using maximal pivotal elements. The solutions in the matrix B are handled in parallel, i.e., each element in a row of B is computed before going on to another row.

The matrix A is destroyed and matrix B contains the solutions to the problem.
| STR  | F3  |  |  |  | SLQ: 36 |
|------|-----|  |  |  |       |
| CAD  | M4  |  |  |  | SLQ: 40 |
| MPY  | COMMON |  |  |  | SLQ: 42 |
| STR  | COMMON |  |  |  | SLQ: 43 |
| NXROW| LFR  | 6, COMMON+M3+M0+4 |  |  | SLQ: 45 |
| ADM  | 8, 1 |  |  |  | SLQ: 46 |
| LFR  | 7, COMMON+M2+M0+5 |  |  |  | SLQ: 49 |
| ADM  | 12, M0+M3-1 |  |  |  | SLQ: 50 |
| CAD  | F3  |  |  |  | SLQ: 51 |
| MPY  | 12, 1 |  |  |  | SLQ: 52 |
| CAM  | 6, M3 |  |  |  | SLQ: 53 |
| STR  | F2  |  |  |  | SLQ: 54 |
| FLD  |  |  |  |  | SLQ: 55 |
| CSB  | 8, 1 |  |  |  | SLQ: 56 |
| MPY  | F2  |  |  |  | SLQ: 57 |
| ASC  | 12, 1 |  |  |  | SLQ: 58 |
| CJF  | 6, 0 |  |  |  | SLQ: 59 |
| CSM  | 6, M1 |  |  |  | SLQ: 60 |
| FLD  |  |  |  |  | SLQ: 62 |
| CSB  | 9, 1 |  |  |  | SLQ: 63 |
| MPY  | F2  |  |  |  | SLQ: 64 |
| ASC  | 13, 1 |  |  |  | SLQ: 65 |
| CJF  | 6, 0 |  |  |  | SLQ: 66 |
| CJU  | 2, 0, NXROW |  |  |  | SLQ: 67 |
| LFR  | 6, COMMON+M3+M0+4 |  |  |  | SLQ: 68 |
| ADM  | 8, 1 |  |  |  | SLQ: 69 |
| CAM  | 2, M3 |  |  |  | SLQ: 70 |
| FLD  |  |  |  |  | SLQ: 71 |
LDM 3, M13
CAM 2, M3
CSM 3, M0
ORDTBL LFR 5, COMMON+M3+M0+5
CAM 6, M2
CAM 7, M2
ADM 2, M1
SFR 5, COMMON+M3+M0+5
CJU 3, 0, ORDTBL
CSM 3, M0-1
CAM 13
ORDER LFR 5, COMMON+M3+M0+4
EOM 6, 3, M5
CAM M8
JZM 8, 0, OMIT
CAM 2, M3
SWITCH LFR 6, COMMON+M2+M0+5
EOM 10, 3, M5
CAM M12
JZM 12, 0, OUT
CJU 2, 0, SWITCH
OUT CAM 10, M6
SFR 6, COMMON+M2+M0+5
CSM 2, M1
FLD
CAD M11
XCH 7, 1,
STR 11, 1,
CJF 2, 0,
ADM 13, 1
OMIT CJU 3, 0, ORDER
CAD COMMON
CRM 13, 1
JPM 13, DETOK
MPY -1.
DETOK LFR 4, COMMON+1
   LFR 5, COMMON+2
   LFR 6, COMMON+3
   LFR 7, COMMON+4
   JLH M3
NAME: Random Number Generator

TYPE: Closed Subroutine

TEMPORARY STORAGE: Internal

FAST REGISTERS CHANGED: Accumulator, FO, Fl

EXECUTION TIME: 25 μsec for 13-bit numbers, 60 μsec for 52-bit numbers

DESCRIPTION: The subroutine will generator unnormalized floating-point random numbers as specified by parameters in MO and Ml. Two types of results are possible. Both are entered by

CALL RANl

1) MO = number of 52-bit numbers to be generated (MO > 0).

Ml = storage location of first number; subsequent numbers are stored sequentially.

2) MO = 0 means generate only a 13-bit number and put it in MO.

Ml is not used here.

Two independent methods of generation are used. One is for the 45-bit fraction, and the other for the 7-bit exponent.

The fractional part is created using a modification of the sequence
DESCRIPTION (Continued): \[ F_N = (F_{N-1} \cdot 5^{19}) \mod 2^{44}, \quad F_0 = 1 \]

with cycle \(2^{42}\).

Experience has shown that there is a noticeable bias in the last few bits of the numbers generated by this sequence. A corrective step consisting of a right shift of four bits is used to ament the anomaly. Detailed testing of this generator is described in File No. 612.\(^1\)

For the exponent part a modified version of the Fibonacci sequence

\[ e_n = (e_{n-1} + e_{n-2}) \mod 2^{13}, \quad e_0 = 0, \quad e_1 = 1 \]

is employed. The change consists of doing a single bit circular left shift of \(e_n\). This modified sequence has a period of 62,445,728. Further details of this generator, as well as a fuller description of the fractional algorithm used above, can be found in File No. 608.\(^2\)

The right-hand seven bits of the shifted version of \(e_n\) are used as the exponent of the 52-bit result, and the left-hand bit is used as the sign. In the case of entry with \(MO = 0\), the 13-bit shifted version of \(e_n\) is used directly.

It will be noted that every time this subroutine is reloaded, the sequences are initialized. If one

\(^1\)Random Number Generator Test Procedures Applied to a Modified, Multiplicative, Congruential Generation Method, by D. K. Chow.

\(^2\)Random Number Generators for ILLIAC II, by Gilbert Cooper.
DESCRIPTION (Continued): wishes to extend the sequence over many runs, the following procedure may be used.

1) After using the subroutine for the last time do a

CALL RANLB

A card will be punched. Save it.

2) The next time the program is loaded do a

CALL RANLC

before the first entry to the random number generator. Include with your program deck a dollar-data card followed by the card punched above.

These two subroutines either punch or read locations RANP and RANP + 1 in eight octal quarter words. The format is

\[ e_{n-1}, e_n, 0, 2, F_n \] (four quarter words)

Inter-subroutine access to RANP is made by a special entry in RANL, namely RANLA.
COMMON
 ENTRY RAN1, RAN1A

RAN1
 BSS 2
 SFR 5, COMMON
 SFR 4, COMMON+1
 LFR 5, RANP
 CSM 0, M0

RANL
 ADM 5, M4
 CAM 4, M5-M4
 CRM 5, 12
 JUM 0, ROX
 CAM 0, M5

FIN
 SFR 5, RANP
 LFR 5, COMMON

ROX
 CAD RANP+2
 M PY RANP+1
 CAE M6
 SAL RANP+1
 LRS M7

SAL F0
 CAD F0
 ANN 5, 127
 CAE M6

ANN 5, 4096
 LOR 0
 SAM 1, 1
 CJU 0, RANL
 LFR 4, COMMON+1

TRA FIN
 FIL
 RANP OCTQ 0,1,2
 OCTQ 3, 200
 OCTQ 261,12127,10275,1200

RANS DEQ 9, RANP, 8, RANF
 RANF CHR 4, 8Q6

RAN1 000
 RAN1 001
 RAN1 002
 RAN1 003
 RAN1 004
 RAN1 005
 RAN1 006
 RAN1 007
 RAN1 008
 RAN1 009
 RAN1 010
 RAN1 011
 RAN1 012
 RAN1 013
 RAN1 014
 RAN1 015
 RAN1 016
 RAN1 017
 RAN1 018
 RAN1 019
 RAN1 020
 RAN1 021
 RAN1 022
 RAN1 023
 RAN1 024
 RAN1 025
 RAN1 026
 RAN1 027
 RAN1 028
 RAN1 029
 RAN1 030
 RAN1 031
 RAN1 032
 RAN1 033
 RAN1 034
RAN1A CAM 1,RANS
     JLH M3
     GO
ENTRY RAN1B,RAN1C

COMMON BSS 1
RAN1B SFR 4,COMMON
     CALL RAN1A
     CALL PUNCH
     LFR 4,COMMON
     JLH M3
     GO
GET PARAMETERS FOR PUNCH-READ
     EXIT

RAN1C SFR 4,COMMON
     CALL RAN1A
     CALL READ
     LFR 4,COMMON
     JLH M3
     GO
SAVE F4
     GO TO PARAMETER FETCH
     PUNCH FOR NEXT TIME
     REPLACE F4
     EXIT

RAN1 035
RAN1 036
RAN1 037
RAN1 038
RAN1 039
RAN1 040
RAN1 041
RAN1 042
RAN1 043
RAN1 044
RAN1 045
RAN1 046
RAN1 047
RAN1 048
RAN1 049
RAN1 050
NAME: Typewriter Or Paper-tape Output System

TYPE: Collection of output subroutines compatible with interrupt mode

TEMPORARY STORAGE: Each subroutine contains its own area of temporary storage.

FAST REGISTERS CHANGED: None

EXECUTION TIME: Variable, depending on output device selected

DESCRIPTION: These subroutines are useful mainly for engineering purposes. The user may specify choice of three modes of operation:

(1) Bypass all output
(2) Output on the on-line IBM Selectric typewriter
(3) Output on paper tape

The subroutines included are listed below with information on their use.

Note: If a print sequence does not begin with one of these subroutines, the programmer should start with

```
CAM 1,48  (See table 1, entry 48)
CALL PTA
```

to assure that the typewriter is in the output mode.
USE:

Choice of Mode:

The first two bits of $SR34_8$ specify mode:

Bit 0 = 0 for output
    = 1 for bypass output
Bit 1 = 0 for typewriter
    = 1 for paper tape

SUBROUTINES INCLUDED:

1. Name: Punch-Type Alternator

   ML = character in paper tape code

   Length: 42 words

   Other subroutines used: None

   Use:
   To output one character in mode specified, set ML and enter via

   CALL PTA

2. Name: Punch or Type Message

   ML = address of first word of message

   Length: 13 words

   Other subroutines used: Number 1

   Use:
   To output a message, set ML and enter via

   CALL PTMSS

   The message should start at a word boundary,
   one character per quarter-word, in paper tape
   code (see table 1). A quarter word of all
   ones is used as the terminator symbol.

3. Name: (A) Punch or Type Quarter Word in Octal

   ML = quarter word

   (B) Punch or Type Full Word in Octal

   ML = address of word

   Length: 31 words

   Other subroutines used: Number 1

   Use:
   (A) To output a quarter word in octal format, set ML and enter via

Date: 7/14/64
Section: 8.4-J6-TOPS
Page: 2 of 39
Change: 2
CALL PTQW  for no preceding character
CALL PTQW1 to precede by a LF/CR
CALL PTQW2 to precede by a space
or CALL PTQW3 to precede by a tab

(B) To output a full word from memory in
octal format (4 quarter words per line)
set M1 and enter via

CALL PTFW  for no preceding character
CALL PTFW1 to precede by a LF/CR
CALL PTFW2 to precede by a space
or CALL PTFW3 to precede by a tab

4. Name:  Punch or Type Decimal Quarter Word
          ML = quarter word

Length:  35 words

Other subroutines used: Number 1

Use:  To output a quarter word as a positive
decimal number, 0 ≤ n ≤ 8191, (4 digits
right justified with leading zeros suppressed) set M1 and enter via

CALL PTDQ  for no preceding character
CALL PTDQ1 to precede by a LF/CR
CALL PTDQ2 to precede by a space
or CALL PTDQ3 to precede by a tab

5. Name:  Punch or Type Full Word in Octal with Address
           ML = address of word

Length:  7 words

Other subroutines used: Number 3, Number 4 (Number 1)

Use:  To output a full word from memory in octal
format with the address as follows:

       100 00144  02760 13500 10202 00137
set M1 and enter via

CALL PTTWA

Date:   11/14/64
Section: 8.4-J6-TOPS
Page:   3 of 39
Change: 2
6. Name: Punch or Type Sexadecimal Word
Length: 40 words
Other subroutines used: Number 1
Use: To output a full word from memory in floating point sexadecimal format, set M1 and enter via

CALL PTSW for no preceding character
CALL PTSW1 to precede by a LF/CR
CALL PTSW2 to precede by a space
or CALL PTSW3 to precede by a tab

SPECIAL ENTRY
This subroutine takes a word from memory with 7-bit exponent and extends the exponent to 8 bits by duplicating the sign bit. If it is desired to output a word from the accumulator with full 8-bit exponent, load the exponent into M1, the word into F5, and enter via

CALL PTSWS

7. Name: Punch or Type Sexadecimal Word with Address
Length: 7 words
Other subroutines used: Number 3, Number 4, Number 6 (Number 1)
Use: To output a full word from memory in floating point sexadecimal format with the address as follows:

100 00144 05d0+042080 bd
set M1 and enter via

CALL PTSWA
8. Name: Punch or Type Decimal Word (with variations)
M1 = some value as specified below

Length: 131 words

Other subroutines used: Number 1

Use:

1. M1 = address of word

To output a full word from memory in floating point decimal format, set M1 and enter via

CALL PTDW for no preceding character
CALL PTDW1 to precede by a LF/CR
CALL PTDW2 to precede by a space
or CALL PTDW3 to precede by a tab

2. M1 = address of parameter word

To obtain a fixed format printout of N consecutive variables starting at location L, use:

CAM 1,PARAM
CALL PTFDW

The format calls for up to eight words on a line, each word of the form

\[ \text{bbbt-00000000000E+00} \]

3 blanks 13 2

PARAM is a location in memory specifying the parameters N and L as

PARAM DECQ N,L,,

3. M1 = not used

To obtain a fixed format printout of the current contents of Amost use:

CALL PTFDA

Date: 11/14/64
Section: 8.4-J6-TOFS
Page: 5 of 39
Change: 2
The contents of AMOST will appear in the format

\[ \pm 0.00000000000000E+00 \]

13 2

4. M1 = address of parameter list

To print the contents of AMOST in a very adaptable manner, use:

```
CAM 1,PARAM
CALL PTFDWA
```

PARAM is the address in core memory of two words containing the following information:

```
PARAM DECQ S,N,K,,F,C,P,R
```

The parameters have the following meanings:

1. S is the number of spaces preceding the sign.
   - S = -1 means 1 LF/CR
   - S = 0 means no space
   - Range: \(-1 \leq S\)

2. N is the total number of digits computed; if leading zeros are printed, N is the total number of digits printed out.
   - Range: \(1 \leq N (\leq 13 \text{ for } F = 1 \text{ only})\).

3. K is the number of digits after the decimal point; K = 0 means no decimal point.
   - Range: \(0 \leq K \leq N\).
4. F specifies the format
   F = 0 gives the format
   \[ \frac{K}{S} +00...00.0...0 \]
   \[ \frac{N}{S} \]
   This format utilizes a feature called automatic N increment (see a. in notes).
   F = 1 gives the format
   \[ \frac{K}{S} +00...00.0...0E+00 \]
   \[ \frac{N}{S} \]
   2

5. C is the desired character to precede the number if positive. All negative numbers are preceded by "-". C must be a decimal code appearing in PTCON.

6. P = 0 means initial zeros are suppressed.
   P = 1 means suppressed initial zeros are replaced as dictated by R (see 7 below). This parameter has effect only for format F = 0.

7. R = 0 means suppressed initial zeros are replaced by nothing.
   R = 1 means suppressed initial zeros are replaced by spaces.

Notes on format 0

a. Automatic N-increment
   If the number to be converted is \( \geq 10^{N-K} \), it cannot be correctly represented by N-K decimal digits before the decimal point.
Hence, \( N \) is increased until the number is less than \( 10^{N-K} \). Notice: There is no \( N \)-increment for format \( F = 1 \).

b. If the user asks for \( K \) decimal digits after the point, the number \( a_0 \) is multiplied by \( 10^K \). If \( 10^K a_0 \geq 2^{64} = 10^{38} \), this leads to overflow and a nonsense result. This limitation of the size of \( K \) is no real restriction, however, since the additional digits after the point would not be significant anyway.

c. The method used to convert a number \( a_0 \) from binary to decimal assumes that \( a_0 \) is an integer. In general, \( a_0 \) is not an integer; but as long as \( |a_0| < 2^{44} \), it is rounded and the integer part \((a_0 + 1/2)\) is converted exactly. If \( |a_0| \geq 2^{44} \), this is no longer possible, and the unrounded \( a_0 \) is submitted to the algorithm.

Registers Destroyed: R, ES

Accuracy: This subroutine is not planned for maximum accuracy; frequent multiplications by 10 may generate a considerable round-off error. Integers up to 13 digits are exact. For format \( F = 0 \), more than 13 digits are not significant.

Acknowledgment: This program is an adaptation of the original paper tape output routine (J3-DPR1-24v) written by Jurg Nievergelt on October 30, 1962, for the Digital Computer Laboratory at the University of Illinois.

Date: 11/14/64
Section: 8.4-J6-TOPS
Page: 8 of 39
Change: 2
9. Name: Punch or Type Decimal Word with Address
ML = address of word

Length: 7 words

Other subroutines used: Number 3, Number 4, Number 8 (Number 1)

Use: To output a fullword from memory in floating point decimal format with the address as follows:

    100 00144 +.1234567890123E+02
    set ML and enter via
    CALL PTDWA

10. Name: Memory Dump Control for Full Word Octal
ML = address of parameter word

Length: 6 words

Other subroutines used: Number 3, Number 16 (Number 1)

Use: To dump a portion of the memory in fullword octal format, set ML and enter via

    CALL PTMDF

The parameter word should contain:

Q. W. 0) not used
Q. W. 1) First word address
Q. W. 2) Last word address
Q. W. 3) not used

11. Name: Memory Dump Control for Full Word Octal with Addresses
ML = address of parameter word (see #10)

Length: 6 words

Other subroutines used: Number 5, Number 16, (Number 1, Number 3, Number 4)
Use: To dump a portion of the memory in full word octal format with addresses, set ML and enter via

CALL PTMFA

12. Name: Memory Dump Control for Sexadecimal
ML = address of parameter word (see #10)
Length: 6 words
Other subroutines used: Number 6, Number 16 (Number 1)
Use: To dump a portion of the memory in floating point sexadecimal format, set ML and enter via

CALL PTMDS

13. Name: Memory Dump Control for Sexadecimal with Addresses
ML = address of parameter word (see #10)
Length: 6 words
Other subroutines used: Number 7, Number 16 (Number 1, Number 3,
Number 4, Number 6)
Use: To dump a portion of the memory in floating point sexadecimal format with addresses, set ML and enter via

CALL PTMSA

14. Name: Memory Dump Control for Full Word Decimal
ML = address of parameter word (see #10)
Length: 6 words
Other subroutines used: Number 8, Number 16 (Number 1)
Use: To dump a portion of the memory in floating point decimal format, set ML and enter via

CALL PTMDD
15. Name: Memory Dump Control for Full Word Decimal with Addresses

ML = address of parameter word (see #10)

Length: 6 words

Other subroutines used: Number 9, Number 16 (Number 8, Number 3, Number 4, Number 1)

Use: To dump a portion of the memory in floating point decimal format with addresses, set ML and enter via

CALL PTDMDA

16. Name: Punch or Type Memory Dump

Length: 9 words

Other subroutines used: None

Use: This subroutine should not be called by the programmer. It is used by subroutines Number 10, Number 11, Number 12, Number 13, Number 14, and Number 15 to produce a memory dump in the specified format.

17. Name: Fast Register Dump Control for Full Word Octal Format

ML = address of storage

Length: 5 words

Other subroutines used: Number 3, Number 20 (Number 1)

Use: To dump fast registers 2 through 7 (properly labeled) in full word octal format, store F4 in memory, set ML to the address of storage, and enter via

CALL PTFRDØ

Date: 11/14/64
Section: 8.4-J5-TOPS
Page: 11 of 39
Change: 2
The original contents of F4 (as stored in memory) are restored to F4 prior to exit.

18. Name: Fast Register Dump Control for Sexadecimal Format
Length: 5 words
Other subroutines used: Number 6, Number 20 (Number 1)
Use: To dump fast registers 2 through 7 in sexadecimal format, proceed as in Number 17, but enter via

        CALL    PTFRDS

19. Name: Fast Register Dump Control for Full Word Decimal Format
Length: 5 words
Other subroutines used: Number 8, Number 20 (Number 1)
Use: To dump fast registers 2 through 7 in decimal format, proceed as in Number 17 but enter via

        CALL    PTFRDD

20. Name: Punch or Type Fast Register Dump
Length: 25 words
Other subroutines used: Number 1
Use: This subroutine should not be called by the programmer. It is used by subroutines Number 17, Number 18, and Number 19 to produce a fast register dump in the specified format.

Date: 11/14/64
Section: 8.4-J6-TOPS
Page: 12 of 39
Change: 2
Punch or Type ACCumulator Dump
M1 = not used

88 words

Number 1, Number 2, Number 6

To dump Amost, Aleast, R,ES, FO(Out), and
Fl(In) as floating point sexadecimal numbers,
enter via

CALL PTACC

This subroutine also indicates whether OV or Z
indicators were on upon entry. All information
is restored except R,ES.

Note that R,ES contains the remainder immediately
after division, but otherwise may be meaningless.
If the first two bits of ES differ, OV will be set
during storage (see EXCEPTIONS below). Should
this occur, the comment "OV set during storage"
will occur in the output, although the true con-
tents of ES will be given.

EXCEPTIONS:

If Z indicator is on or if R contains zero, the
setting of OV during storage of R,ES will be in-
hibited. Therefore, if either of the two conditions
are true, a question mark will appear after ES to
indicate that the true value of the first bit of ES
is undeterminable by a program.
<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Octal Equivalent</th>
<th>Paper Tape Code</th>
<th>Typewriter Character Assigned</th>
<th>Decimal Value</th>
<th>Octal Equivalent</th>
<th>Paper Tape Code</th>
<th>Typewriter Character Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000000</td>
<td>0</td>
<td>0</td>
<td>64</td>
<td>001000</td>
<td>)</td>
<td>)</td>
</tr>
<tr>
<td>1</td>
<td>000001</td>
<td>1</td>
<td>1</td>
<td>65</td>
<td>001010</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>2</td>
<td>000002</td>
<td>2</td>
<td>2</td>
<td>66</td>
<td>001020</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>3</td>
<td>000003</td>
<td>3</td>
<td>3</td>
<td>67</td>
<td>001030</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>4</td>
<td>000004</td>
<td>4</td>
<td>4</td>
<td>68</td>
<td>001040</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>5</td>
<td>000005</td>
<td>5</td>
<td>5</td>
<td>69</td>
<td>001050</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>000006</td>
<td>6</td>
<td>6</td>
<td>70</td>
<td>001060</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>7</td>
<td>000007</td>
<td>7</td>
<td>7</td>
<td>71</td>
<td>001070</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>8</td>
<td>000010</td>
<td>8</td>
<td>8</td>
<td>72</td>
<td>001100</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>9</td>
<td>000011</td>
<td>9</td>
<td>9</td>
<td>73</td>
<td>001110</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>10</td>
<td>000012</td>
<td>+</td>
<td>+</td>
<td>74</td>
<td>001120</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>11</td>
<td>000013</td>
<td>-</td>
<td>-</td>
<td>75</td>
<td>001130</td>
<td>(</td>
<td>(</td>
</tr>
<tr>
<td>12</td>
<td>000014</td>
<td>a</td>
<td>a</td>
<td>76</td>
<td>001140</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>13</td>
<td>000015</td>
<td>b</td>
<td>b</td>
<td>77</td>
<td>001150</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>14</td>
<td>000016</td>
<td>c</td>
<td>c</td>
<td>78</td>
<td>001160</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>15</td>
<td>000017</td>
<td>d</td>
<td>d</td>
<td>79</td>
<td>001170</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>16</td>
<td>000020</td>
<td>e</td>
<td>e</td>
<td>80</td>
<td>001200</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>17</td>
<td>000021</td>
<td>f</td>
<td>f</td>
<td>81</td>
<td>001210</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>18</td>
<td>000022</td>
<td>g</td>
<td>g</td>
<td>82</td>
<td>001220</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>19</td>
<td>000023</td>
<td>h</td>
<td>h</td>
<td>83</td>
<td>001230</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>20</td>
<td>000024</td>
<td>i</td>
<td>i</td>
<td>84</td>
<td>001240</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>21</td>
<td>000025</td>
<td>j</td>
<td>j</td>
<td>85</td>
<td>001250</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>22</td>
<td>000026</td>
<td>k</td>
<td>k</td>
<td>86</td>
<td>001260</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>23</td>
<td>000027</td>
<td>l</td>
<td>l</td>
<td>87</td>
<td>001270</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>24</td>
<td>000030</td>
<td>m</td>
<td>m</td>
<td>88</td>
<td>001300</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>25</td>
<td>000031</td>
<td>n</td>
<td>n</td>
<td>89</td>
<td>001310</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>26</td>
<td>000032</td>
<td>o</td>
<td>o</td>
<td>90</td>
<td>001320</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>27</td>
<td>000033</td>
<td>p</td>
<td>p</td>
<td>91</td>
<td>001330</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>28</td>
<td>000034</td>
<td>q</td>
<td>q</td>
<td>92</td>
<td>001340</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>29</td>
<td>000035</td>
<td>r</td>
<td>r</td>
<td>93</td>
<td>001350</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>30</td>
<td>000036</td>
<td>s</td>
<td>s</td>
<td>94</td>
<td>001360</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>31</td>
<td>000037</td>
<td>t</td>
<td>t</td>
<td>95</td>
<td>001370</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
### TABLE 1. PTCON, Punch to Type CONversion (Continued)

<table>
<thead>
<tr>
<th>Decimal Value</th>
<th>Octal Equivalent</th>
<th>Paper Tape Code</th>
<th>Typewriter Character Assigned</th>
<th>Decimal Value</th>
<th>Octal Equivalent</th>
<th>Paper Tape Code</th>
<th>Typewriter Character Assigned</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>00040</td>
<td>u</td>
<td>u</td>
<td>96</td>
<td>00140</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>33</td>
<td>00041</td>
<td>v</td>
<td>v</td>
<td>97</td>
<td>00141</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>34</td>
<td>00042</td>
<td>w</td>
<td>w</td>
<td>98</td>
<td>00142</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>35</td>
<td>00043</td>
<td>x</td>
<td>x</td>
<td>99</td>
<td>00143</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>36</td>
<td>00044</td>
<td>y</td>
<td>y</td>
<td>100</td>
<td>00144</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>37</td>
<td>00045</td>
<td>z</td>
<td>z</td>
<td>101</td>
<td>00145</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>38</td>
<td>00046</td>
<td></td>
<td></td>
<td>102</td>
<td>00146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>00047</td>
<td></td>
<td></td>
<td>103</td>
<td>00147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>00050</td>
<td></td>
<td></td>
<td>104</td>
<td>00150</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>00051</td>
<td></td>
<td></td>
<td>105</td>
<td>00151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>00052</td>
<td></td>
<td>NOP</td>
<td>106</td>
<td>00152</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>43</td>
<td>00053</td>
<td></td>
<td>NOP</td>
<td>107</td>
<td>00153</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>44</td>
<td>00054</td>
<td></td>
<td>NOP</td>
<td>108</td>
<td>00154</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>45</td>
<td>00055</td>
<td></td>
<td>NOP</td>
<td>109</td>
<td>00155</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>46</td>
<td>00056</td>
<td></td>
<td>NOP</td>
<td>110</td>
<td>00156</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>47</td>
<td>00057</td>
<td></td>
<td>NOP</td>
<td>111</td>
<td>00157</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>48</td>
<td>00060</td>
<td>delay</td>
<td>S.F.O.M.</td>
<td>112</td>
<td>00160</td>
<td>tab</td>
<td>NOP</td>
</tr>
<tr>
<td>49</td>
<td>00061</td>
<td></td>
<td>NOP</td>
<td>113</td>
<td>00161</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>50</td>
<td>00062</td>
<td></td>
<td>black</td>
<td>114</td>
<td>00162</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>51</td>
<td>00063</td>
<td></td>
<td>NOP</td>
<td>115</td>
<td>00163</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>52</td>
<td>00064</td>
<td></td>
<td>NOP</td>
<td>116</td>
<td>00164</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>53</td>
<td>00065</td>
<td></td>
<td>NOP</td>
<td>117</td>
<td>00165</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>54</td>
<td>00066</td>
<td></td>
<td>NOP</td>
<td>118</td>
<td>00166</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>55</td>
<td>00067</td>
<td></td>
<td>NOP</td>
<td>119</td>
<td>00167</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>56</td>
<td>00070</td>
<td></td>
<td>space</td>
<td>120</td>
<td>00170</td>
<td>backsp</td>
<td>NOP</td>
</tr>
<tr>
<td>57</td>
<td>00071</td>
<td>clr tab</td>
<td>NOP</td>
<td>121</td>
<td>00171</td>
<td>set tab</td>
<td>NOP</td>
</tr>
<tr>
<td>58</td>
<td>00072</td>
<td>clr all</td>
<td>NOP</td>
<td>122</td>
<td>00172</td>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>59</td>
<td>00073</td>
<td></td>
<td>NOP</td>
<td>123</td>
<td>00173</td>
<td></td>
<td>NOP</td>
</tr>
<tr>
<td>60</td>
<td>00074</td>
<td></td>
<td>HALD index</td>
<td>124</td>
<td>00174</td>
<td>HLU</td>
<td>NOP</td>
</tr>
<tr>
<td>61</td>
<td>00075</td>
<td></td>
<td>NOP</td>
<td>125</td>
<td>00175</td>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>62</td>
<td>00076</td>
<td></td>
<td>*</td>
<td>126</td>
<td>00176</td>
<td>$</td>
<td>NOP</td>
</tr>
<tr>
<td>63</td>
<td>00077</td>
<td>LF/CR</td>
<td>LF/CR</td>
<td>127</td>
<td>00177</td>
<td>erase</td>
<td>NOP</td>
</tr>
</tbody>
</table>

8191 17777 special character used as the terminator for PIMSS

* This assignment is made in order to have a character which will set the typewriter for output mode without affecting paper tape output.

Date: 11/14/64
Section: 8.4-J6-TOPS
Page: 15 of 39
Change: 2
### TYPEWRITER OR PAPER-TAPE OUTPUT SYSTEM: (J6-TOPS-01/UI-AL)

#### NUMBER 1) PUNCH-TYPE ALTERNATOR

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>PTA</th>
<th>TOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FREE F4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>READ SR34</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FREE F7</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FETCH PROPER WORD OF PTCON</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTEMP</th>
<th>BSS 2</th>
<th>TOPS 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4152, 4217, 4218, 4155, 4184, 4121, 4122, 4187</td>
<td>01234567</td>
</tr>
<tr>
<td></td>
<td>4220, 4157, 4128, 4223, 4098, 4163, 4196, 4133</td>
<td>89+--ABCD</td>
</tr>
<tr>
<td></td>
<td>4134, 4199, 4100, 4165, 4166, 4103, 4208, 4145</td>
<td>EFGHIJKL</td>
</tr>
<tr>
<td></td>
<td>4146, 4211, 4112, 4177, 4178, 4115, 4148, 4213</td>
<td>MNOPQRSTUVWXYZ</td>
</tr>
<tr>
<td></td>
<td>4214, 4151, 4180, 4117, 4118, 4183, 4097, 4348</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4286, 4158,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3840, . . . .</td>
<td></td>
</tr>
<tr>
<td></td>
<td>256, . . . , 1024, 4194, 768</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>4259, 4250, 4285, 4280, 4345, 4346, 4249, 4321</td>
<td>)</td>
</tr>
<tr>
<td></td>
<td>4351, 4322, 4193, 4283, 4226, 4291, 4324, 4261</td>
<td>(= ABCD</td>
</tr>
<tr>
<td></td>
<td>4262, 4327, 4228, 4293, 4294, 4231, 4336, 4273</td>
<td>EFGHIJKL</td>
</tr>
<tr>
<td></td>
<td>4274, 4339, 4240, 4305, 4203, 4243, 4276, 4341</td>
<td>MNOPQRSTUVWXYZ</td>
</tr>
<tr>
<td></td>
<td>4342, 4279, 4308, 4245, 4246, 4311, 4225, 4262</td>
<td>U VWXYZ E</td>
</tr>
<tr>
<td></td>
<td>4131, 4160,</td>
<td>/ ,</td>
</tr>
<tr>
<td></td>
<td>1280, . . . .</td>
<td></td>
</tr>
<tr>
<td></td>
<td>512, . . . , 4256,</td>
<td>$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PTA</th>
<th>SFR 4, PTEMP</th>
<th>TOPS 17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FREE F4</td>
<td>TOPS 18</td>
</tr>
<tr>
<td></td>
<td>READ SR34</td>
<td>TOPS 19</td>
</tr>
<tr>
<td></td>
<td>FREE F7</td>
<td>TOPS 20</td>
</tr>
<tr>
<td></td>
<td>FETCH PROPER WORD OF PTCON</td>
<td>TOPS 21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SFR</th>
<th>FREE F4</th>
<th>TOPS 18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FREE F7</td>
<td>TOPS 20</td>
</tr>
<tr>
<td></td>
<td>FETCH PROPER WORD OF PTCON</td>
<td>TOPS 21</td>
</tr>
<tr>
<td></td>
<td>FREE F7</td>
<td>TOPS 22</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JNM</th>
<th>2, PTA1</th>
<th>TOPS 18</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BYPASS OUTPUT</td>
<td>TOPS 19</td>
</tr>
<tr>
<td></td>
<td>FREE F7</td>
<td>TOPS 20</td>
</tr>
<tr>
<td></td>
<td>FETCH PROPER WORD OF PTCON</td>
<td>TOPS 21</td>
</tr>
<tr>
<td></td>
<td>FREE F7</td>
<td>TOPS 22</td>
</tr>
<tr>
<td></td>
<td>FETCH PROPER WORD OF PTCON</td>
<td>TOPS 23</td>
</tr>
</tbody>
</table>

| LFR   | 7, PCON      | TOPS 18 |

---

**Date:** 8/4-7/73

**Change:** 2 of 39
OR8    M1                     TYPE PROPER CHARACTER
ATN    M12
SSR    6                     RESTORE F7
LFR    7,PTEMP+1             RESTORE F4
PTA1   LFR                   EXIT
      4,PTEMP
JLH    M3                     GO TO EXIT
PTA2   ATN                   PUNCH CHARACTER
      M1
SSR    PTA1
TRA    GO                     TOPS 30
      TOPS 31
      TOPS 32
      TOPS 33
      TOPS 34
      TOPS 35
      TOPS 36
      TOPS 37
      TOPS 38
      TOPS 39
      TOPS 40
      TOPS 41
      TOPS 42
      TOPS 43
      TOPS 44
      TOPS 45
      TOPS 46
      TOPS 47
      TOPS 48
      TOPS 49
      TOPS 50
      TOPS 51
      TOPS 52
      TOPS 53
      TOPS 54
      TOPS 55
      TOPS 56
      TOPS 57
      TOPS 58
      TOPS 59
      TOPS 60
      TOPS 61

***: NUMBER 21: PUNCH OR TYPE MESSAGE

ENTRY   PTMSS
PTMSS1  BSS     2
PTMSS   SFR     4,PTEMP1
            7,PTEMP1+1
CAM     ,M1
CAM     1,48
CNM     2
TRA     PTMSS3
PTMSS2  CSM     2,4
ATN     1,1
LFR     7
PTMSS   ORB     M2
            1,M12+1
CAM     1,PTMSS4
JZM     1
SBM     1,1
PTMSS3  CALL    PTA
            PTMSS1
            PTMSS2
            CJU    2,PTMSS2
CTA     PTMSS1
LFR     7,PTEMP1+1
LFR     4,PTEMP1
JLH     M3
GO
TOPS 40
TOPS 41
TOPS 42
TOPS 43
TOPS 44
TOPS 45
TOPS 46
TOPS 47
TOPS 48
TOPS 49
TOPS 50
TOPS 51
TOPS 52
TOPS 53
TOPS 54
TOPS 55
TOPS 56
TOPS 57
TOPS 58
TOPS 59
TOPS 60
TOPS 61
### NUMBER 3A) PUNCH OR TYPE QUARTER WORD IN OCTAL

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>PTEFW, PTEFW1, PTEFW2, PTEFW3, PTQW, PTQW1, PTQW2, PTQW3</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTEMP2</td>
<td>BSS 3</td>
</tr>
<tr>
<td>PTQW</td>
<td>SFR 4, PTEMP2+2</td>
</tr>
<tr>
<td>CAM</td>
<td>1,48</td>
</tr>
<tr>
<td>TRA</td>
<td>PTQW5</td>
</tr>
<tr>
<td>CAN</td>
<td>0,0</td>
</tr>
<tr>
<td>PTQW1</td>
<td>SFR 4, PTEMP2+2</td>
</tr>
<tr>
<td>CAM</td>
<td>2,63</td>
</tr>
<tr>
<td>TRA</td>
<td>PTQW4</td>
</tr>
<tr>
<td>CAN</td>
<td>0,0</td>
</tr>
<tr>
<td>PTQW2</td>
<td>SFR 4, PTEMP2+2</td>
</tr>
<tr>
<td>CAM</td>
<td>2,56</td>
</tr>
<tr>
<td>TRA</td>
<td>PTQW4</td>
</tr>
<tr>
<td>CAN</td>
<td>0,0</td>
</tr>
<tr>
<td>PTQW3</td>
<td>SFR 4, PTEMP2+2</td>
</tr>
<tr>
<td>CAM</td>
<td>2,112</td>
</tr>
<tr>
<td>PTQW4</td>
<td>CAM 1,48</td>
</tr>
<tr>
<td>CALL</td>
<td>PTA</td>
</tr>
<tr>
<td>CAM</td>
<td>1, M2</td>
</tr>
<tr>
<td>PTQW5</td>
<td>CALL PTA</td>
</tr>
<tr>
<td>LDM</td>
<td>1, PTEMP2+2</td>
</tr>
<tr>
<td>CRM</td>
<td>1,12</td>
</tr>
<tr>
<td>CAM</td>
<td>2</td>
</tr>
<tr>
<td>ANN</td>
<td>2,1</td>
</tr>
<tr>
<td>CAM</td>
<td>1</td>
</tr>
<tr>
<td>CSM</td>
<td>.5</td>
</tr>
<tr>
<td>TRA</td>
<td>PTQW7</td>
</tr>
</tbody>
</table>

Set up special character

Free F4

Needed for spacing

Set up "LF/CR"

Set up "SPACE"

Needed for spacing

Free F4

Set up "TAB"

Set up special character

Go to output character

Fetch next character

Go to output character

Fetch quarter-word

Load shifted quarter-word into M2

Fetch sign bit

Set loop counter

Enter loop
PTQW6  CRM    2,10       SHIFT FOR NEXT CHARACTER
ANN    2,7       LOAD CHARACTER INTO M1
CAM    1

PTQW7  CALL    PTA       GO TO OUTPUT CHARACTER
       CJU     ,PTQW6      MORE CHARACTERS
       LFR     4,PTEMP2+2  RESTORE F4
       JLH     M3         EXIT

*** NUMBER 3B) PUNCH OR TYPE FULL WORD IN OCTAL

PTFW  SFR    4,PTEMP2   FREE F4
       CAM    2          SET RELATIVIZER FOR BLANK
       TRA    PTFW4

PTFW1 SFR    4,PTEMP2   FREE F4
       CAM    2,2        SET RELATIVIZER FOR 'LF/CR'
       TRA    PTFW4

PTFW2 SFR    4,PTEMP2   FREE F4
       CAM    2,4        SET RELATIVIZER FOR 'SPACE'
       TRA    PTFW4

PTFW3 SFR    4,PTEMP2   FREE F4
       CAM    2,6        SET RELATIVIZER FOR 'TAB'

PTFW4 SFR    7,PTEMP2+1 FREE F7
       LFR    7,M1       FETCH WORD
       CSM     4          SET QUARTER-WORD COUNTER

PTFW5 ORB    M0        LOAD PROPER QUARTER-WORD
       CAM    1,M12
       ATN    M2
       CALL   PTQW
       CAM    2,4
       CJU     ,PTFW5
       LFR    7,PTEMP2+1  MORE QUARTER-WORDS TO GO
       LFR    4,PTEMP2   RESTORE F7
       JLH     M3         RESTORE F4
       GO

       EXIT

       TOPS  95
       TOPS  94
       TOPS  93
       TOPS  92
       TOPS  91
       TOPS  90
       TOPS  89

       TOPS  104
       TOPS  103
       TOPS  102
       TOPS  101
       TOPS  100
       TOPS  99
       TOPS  98
       TOPS  97
       TOPS  96

       TOPS  111
       TOPS  110
       TOPS  109
       TOPS  108
       TOPS  107
       TOPS  106
       TOPS  105
       TOPS  104
       TOPS  103
       TOPS  102
       TOPS  101
       TOPS  100
       TOPS  99
       TOPS  98
       TOPS  97
       TOPS  96

       TOPS  119
       TOPS  118
       TOPS  117
       TOPS  116
       TOPS  115
       TOPS  114
       TOPS  113
       TOPS  112
       TOPS  111
       TOPS  110
       TOPS  109
       TOPS  108
       TOPS  107
       TOPS  106
       TOPS  105
       TOPS  104
       TOPS  103
       TOPS  102
       TOPS  101
       TOPS  100
       TOPS  99
       TOPS  98
       TOPS  97
       TOPS  96

       TOPS  119
       TOPS  118
       TOPS  117
       TOPS  116
       TOPS  115
       TOPS  114
       TOPS  113
       TOPS  112
       TOPS  111
       TOPS  110
       TOPS  109
       TOPS  108
       TOPS  107
       TOPS  106
       TOPS  105
       TOPS  104
       TOPS  103
       TOPS  102
       TOPS  101
       TOPS  100
       TOPS  99
       TOPS  98
       TOPS  97
       TOPS  96
### NUMBER 4) PUNCH OR TYPE DECIMAL QUARTER WORD

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>OPERANDS</th>
<th>TOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTEQP3</td>
<td>BSS 3</td>
<td>120</td>
</tr>
<tr>
<td>PZER0</td>
<td>DECQ</td>
<td>121</td>
</tr>
<tr>
<td>PTDOQ</td>
<td>SFR 4,PTEQP3,PTDOQ1,PTDOQ2,PTDOQ3</td>
<td>122</td>
</tr>
<tr>
<td>CAM</td>
<td>1,48</td>
<td>123</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDOQY</td>
<td>124</td>
</tr>
<tr>
<td>PTDOQ1</td>
<td>SFR 4,PTEQP3,PTDOQ1,PTDOQ2,PTDOQ3</td>
<td>125</td>
</tr>
<tr>
<td>CAM</td>
<td>2,63</td>
<td>126</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDOQY</td>
<td>127</td>
</tr>
<tr>
<td>PTDOQ2</td>
<td>SFR 4,PTEQP3,PTDOQ1,PTDOQ2,PTDOQ3</td>
<td>128</td>
</tr>
<tr>
<td>CAM</td>
<td>2,56</td>
<td>129</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDOQY</td>
<td>128</td>
</tr>
<tr>
<td>PTDOQ3</td>
<td>SFR 4,PTEQP3,PTDOQ1,PTDOQ2,PTDOQ3</td>
<td>129</td>
</tr>
<tr>
<td>CAM</td>
<td>2,112</td>
<td>130</td>
</tr>
<tr>
<td>CALL</td>
<td>PTADQY</td>
<td>131</td>
</tr>
<tr>
<td>CAM</td>
<td>1,48</td>
<td>132</td>
</tr>
<tr>
<td>CALL</td>
<td>PTA</td>
<td>133</td>
</tr>
<tr>
<td>CAM</td>
<td>1,2</td>
<td>134</td>
</tr>
<tr>
<td>CALL</td>
<td>PTADY</td>
<td>135</td>
</tr>
<tr>
<td>SFR</td>
<td>7,PTEQP3+1</td>
<td>136</td>
</tr>
<tr>
<td>LDM</td>
<td>1,PTEQP3</td>
<td>137</td>
</tr>
<tr>
<td>LFR</td>
<td>7,PZER0</td>
<td>138</td>
</tr>
<tr>
<td>JPM</td>
<td>1,PTDOQ1</td>
<td>139</td>
</tr>
<tr>
<td>SBM</td>
<td>1,4100</td>
<td>140</td>
</tr>
<tr>
<td>JPM</td>
<td>1,PTDOQ4</td>
<td>141</td>
</tr>
<tr>
<td>CAM</td>
<td>12,4</td>
<td>142</td>
</tr>
<tr>
<td>CAM</td>
<td>14,9</td>
<td>143</td>
</tr>
<tr>
<td>CAM</td>
<td>15,M1+10</td>
<td>144</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDOQK</td>
<td>145</td>
</tr>
</tbody>
</table>

Date: 8/4/76-TOP5
Change: 2/39
PTDQ4 CAM 12,4 CORRECT FOR N G. T. 4099 TOPS 148
CAM 13,1
PTDQA SBM 1,1000 EXTRACT TOPS 149
JNM 1,PTDQB TOPS 150
CJU 12,PTDQA DIGIT TOPS 151
PTDQB ADM 1,1000 TOPS 152
JNM 1,PTDQue TOPS 153
CJU 13,PTDQC DIGIT TOPS 154
PTDQC SBM 1,100 EXTRACT TOPS 155
JNM 1,PTDQD TOPS 156
CJU 13,PTDQC DIGIT TOPS 157
PTDQD ADM 1,100 TOPS 158
JNM 1,PTDQF TOPS 159
CJU 14,PTDQE DIGIT TOPS 160
PTDQE SBM 1,10 EXTRACT UNITS DIGIT TOPS 161
JNM 1,PTDQF TOPS 162
CJU 14,PTDQE DIGIT TOPS 163
PTDQF ADM 15,M1+10 EXTRACT FOR CARRY TOPS 164
SBN 13,10 CORRECT FOR CARRY TOPS 165
CAM 1 TOPS 166
JUM 1,PTDQ5 SCAN TO ELIMINATE TOPS 167
CAM 13 TOPS 168
CJU 12,PTDQK LEADING ZEROS TOPS 169
PTDQ5 JUM 12,PTDQK TOPS 170
CAM 12,56 TOPS 171
JUM 13,PTDQK TOPS 172
CAM 13,56 TOPS 173
JUM 14,PTDQK SET DIGIT COUNTER TOPS 174
CAM 14,56 OUTPUT
PTDQK CSM 2,4 ALL
CAM 1,M12 DIGITS
CALL PTA
CJU 2,PTDQ6
LFR 7,PTEMP3+1
LFR 4,PTEMP3
JLH M3 RESTORE F7
GO EXIT
PTDQ6 ORB M2
CAM 1,M12 RESTORE F4
CALL PTA
CJU 2,PTDQ6 TOPS 175
LFR 7,PTEMP3+1 TOPS 176
LFR 4,PTEMP3 TOPS 177
JLH M3 TOPS 178
GO TOPS 179
PTDQ6 M3 TOPS 180
GO TOPS 181
*** NUMBER 5) PUNCH OR TYPE FULL WORD IN OCTAL WITH ADDRESS

ENTRY PTFWA
PTMP2A CALL PTFW2 NEEDED TO SET UP TRANSFER VECTOR
PTFWA SFR 4,PTMP2A FREE F4
CALL PTQW2
CALL PTQ01 GO TO OUTPUT ADDRESS
LFR 4,PTMP2A RESTORE F4
TRA PTFW2 GO TO OUTPUT WORD
GO

*** NUMBER 6) PUNCH OR TYPE SEXADECIMAL WORD

ENTRY PTSW,PTSW1,PTSW2,PTSW3,PTSW4
PTEMP4 BSS 3
PTSW5 SFR 4,PTEMP4 FREE F4
SFR 5,PTEMP4+1 FREE F5
SFR 7,PTEMP4+2 FREE F7
CAM 15,M1 SAVE EXPONENT
CAM 1,48 SET UP SPECIAL CHARACTER
CALL PTA GO TO OUTPUT CHARACTER
TRA PTSW7
PTSW SFR 4,PTEMP4 FREE F4
CAM 1,48 SET UP SPECIAL CHARACTER
TRA PTSW5
PTSW1 SFR 4,PTEMP4 FREE F4
CAM 2,63 SET UP 'LF/CR'
TRA PTSW4
PTSW2 SFR 4,PTEMP4 FREE F4
CAM 2,56 SET UP 'SPACE'
TRA PTSW4
PTSW3 SFR 4,PTEMP4 FREE F4
CAM 2,112 SET UP 'TAB'
PTSW4 CAM 1,48 SET UP SPECIAL CHARACTER

TOPS 182
TOPS 183
TOPS 184
TOPS 185
TOPS 186
TOPS 187
TOPS 188
TOPS 189
TOPS 190
TOPS 191
TOPS 192
TOPS 193
TOPS 194
TOPS 195
TOPS 196
TOPS 197
TOPS 198
TOPS 199
TOPS 200
TOPS 201
TOPS 202
TOPS 203
TOPS 204
TOPS 205
TOPS 206
TOPS 207
TOPS 208
TOPS 209
TOPS 210
CALL PTA
CAM 1, M2
PTSW5 CALL PTA
SFR 5, PTEMP+4+1
SFR 7, PTEMP+4+2
LDM 1, PTEMP4
LFR 5, M1
CAM 15, M7
CRN 15, 7
CAM 14
JNM 14, PTSW6
ANM 15, 127
TRA PTSW7
PTSW6 ORM 15, 128
PTSW7 CRM 4, 12
ANN 4, 1
CAM 1
CNM 12
CSM 13, 3
CSM 14, 4
TRA PTSW10
PTSW8 CSM 14, 3
PTSW9 CRM 4, 9
ANN 4, 15
CAM 1
PTSW10 CALL PTA
CJU 14, PTSW9
CJZ 13, PTSW11
CAM 4, M5
CJZ 12, PTSW8
ANM 4, 1
ANN 6, 8190
ADM 4
CRM 4, 1

GO TO OUTPUT CHARACTER
FETCH NEXT CHARACTER
GO TO OUTPUT CHARACTER
FREE F5
FREE F7
FETCH ADDRESS
LOAD WORD
LOAD EXPONENT
EXTEND EXPONENT TO 8 BITS
LOAD SIGN BIT
SET GROUP COUNTERS
SET DIGIT COUNTER
ENTER LOOP
SET DIGIT COUNTER
LOAD NEXT DIGIT
GO TO OUTPUT CHARACTER
MORE DIGITS TO GO
3 GROUPS PUNCHED
LOAD GROUP 2
1 GROUP PUNCHED
LOAD GROUP 3
(8190 = 17776)
<table>
<thead>
<tr>
<th>PTSW11</th>
<th>ANM</th>
<th>6,3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANM</td>
<td>7,8064</td>
<td></td>
</tr>
<tr>
<td>ADM</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>CRM</td>
<td>6,2</td>
<td></td>
</tr>
<tr>
<td>CSM</td>
<td>14,2</td>
<td></td>
</tr>
<tr>
<td>PTSW12</td>
<td>CRM</td>
<td>6,9</td>
</tr>
<tr>
<td>ANN</td>
<td>6,15</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**PTSW13**

**CALL PTA**

**GO TO OUTPUT CHARACTER**

**CALL PTW12**

**MORE DIGITS TO GO**

**JUM 13, PTEX5**

**END SIGNAL SET**

**CAM 1,56**

**SET UP 'SPACE'**

**CRN 15,8**

**LOAD EXPONENT**

**CAM 6**

**CSM 14,3**

**SET DIGIT COUNTER**

**PTEX5 LFR 7,PTMP4+2**

**ENTER LOOP AND SET END SIGNAL**

**PTMP4 LFR 5,PTMP4+1**

**RESTORE F5**

**LFR 4,PTMP4**

**RESTORE F4**

**JLH M3**

**EXIT**

**GO**

---

***** NUMBER 7: PUNCH OR TYPE SEHERDECIMAL WORD WITH ADDRESS***

**ENTRY PTSWA**

**PTMP4 CALL PTSW2**

**NEEDED TO SET UP TRANSFER VECTOR**

**PTSWA SFR 4,PTMP4A**

**FREE F4**

**CALL PTDQ1**

**GO TO OUTPUT ADDRESS**

**CALL PTQW2**

**LFR 4,PTMP4A**

**RESTORE F4**

**TRA PTSW2**

**GO TO OUTPUT WORD**

**GO**

---

**TOPS 245**

**TOPS 246**

**TOPS 247**

**TOPS 248**

**TOPS 249**

**TOPS 250**

**TOPS 251**

**TOPS 252**

**TOPS 253**

**TOPS 254**

**TOPS 255**

**TOPS 256**

**TOPS 257**

**TOPS 258**

**TOPS 259**

**TOPS 260**

**TOPS 261**

**TOPS 262**

**TOPS 263**

**TOPS 264**

**TOPS 265**

**TOPS 266**

**TOPS 267**

**TOPS 268**

**TOPS 269**

**TOPS 270**

**TOPS 271**

**TOPS 272**

**TOPS 273**

**TOPS 274**
**Number 8) Punch or Type Decimal Word (with Variations)**

<table>
<thead>
<tr>
<th>ENTRY</th>
<th>PTDW, PTDW1, PTDW2, PTDW3</th>
<th>TOPS 275</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRY</td>
<td>PTFDW, PTFDA, PTFDWA</td>
<td>TOPS 276</td>
</tr>
<tr>
<td>PTEMP8</td>
<td>BSS</td>
<td>15</td>
</tr>
<tr>
<td>PTDW</td>
<td>SFR</td>
<td>4, PTEMP8+1</td>
</tr>
<tr>
<td>CAM</td>
<td>1, 48</td>
<td>SET UP SPECIAL CHARACTER</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDW5</td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>0, 0</td>
<td>NEEDED FOR SPACING</td>
</tr>
<tr>
<td>PTDW1</td>
<td>SFR</td>
<td>4, PTEMP8+1</td>
</tr>
<tr>
<td>CAM</td>
<td>2, 63</td>
<td>SET UP 'LF/CR'</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDW4</td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>0, 0</td>
<td>NEEDED FOR SPACING</td>
</tr>
<tr>
<td>PTDW2</td>
<td>SFR</td>
<td>4, PTEMP8+1</td>
</tr>
<tr>
<td>CAM</td>
<td>2, 56</td>
<td>SET UP 'SPACE'</td>
</tr>
<tr>
<td>TRA</td>
<td>PTDW4</td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>0, 0</td>
<td>NEEDED FOR SPACING</td>
</tr>
<tr>
<td>PTDW3</td>
<td>SFR</td>
<td>4, PTEMP8+1</td>
</tr>
<tr>
<td>CAM</td>
<td>2, 112</td>
<td>SET UP 'TAB'</td>
</tr>
<tr>
<td>PTDW4</td>
<td>CALL</td>
<td>PTA</td>
</tr>
<tr>
<td>CALL</td>
<td>PTA</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>1, M2</td>
<td>FETCH NEXT CHARACTER</td>
</tr>
<tr>
<td>PTDW5</td>
<td>PTA</td>
<td>GO TO OUTPUT CHARACTER</td>
</tr>
<tr>
<td>SFR</td>
<td>6, PTEMP8</td>
<td>SAVE F6</td>
</tr>
<tr>
<td>SFR</td>
<td>7, PTEMP8+5</td>
<td>SAVE F7</td>
</tr>
<tr>
<td>LDM</td>
<td>1, PTEMP8+1</td>
<td>RESTORE M1</td>
</tr>
<tr>
<td>CSM</td>
<td>0, 4</td>
<td>SET ENTRY FLAG TO 4</td>
</tr>
<tr>
<td>TRA</td>
<td>REL6</td>
<td>PROCEED</td>
</tr>
<tr>
<td>PTFDW</td>
<td>SFR</td>
<td>6, PTEMP8</td>
</tr>
<tr>
<td>SFR</td>
<td>7, PTEMP8+5</td>
<td>SAVE F7</td>
</tr>
<tr>
<td>SFR</td>
<td>4, PTEMP8+1</td>
<td>SAVE F4</td>
</tr>
<tr>
<td>CAM</td>
<td>1, 48</td>
<td>SET FOR</td>
</tr>
<tr>
<td>CALL</td>
<td>PTA</td>
<td>OUTPUT MODE</td>
</tr>
<tr>
<td>LDM</td>
<td>1, PTEMP8+1</td>
<td>RESTORE M1</td>
</tr>
</tbody>
</table>
CSM  0,3  SET ENTRY FLAG TO 3
TRA  REL6  PROCEED
PTFDA SFR  6,PTEMP8  SAVE F6
SFR  7,PTEMP8+5  SAVE F7
SFR  4,PTEMP8+1  SAVE F4
CAM  1,48  SET FOR
CALL PTA  OUTPUT MODE
LDM  1,PTEMP8+1  RESTORE M1
CSM  0,2  SET ENTRY FLAG TO 2
TRA  REL6  PROCEED
PTFDWA SFR  6,PTEMP8  SAVE F6
SFR  7,PTEMP8+5  SAVE F7
LFR  6,M1  LOAD PARAMETER WORD
SFR  4,PTEMP8+1  SAVE F4
CAM  1,48  SET FOR
CALL PTA  OUTPUT MODE
LDM  1,PTEMP8+1  RESTORE M1
REL6 SFR  3,PTEMP8+2  SAVE F3
SFR  5,PTEMP8+3  SAVE F5
SFR  4,PTEMP8+14  SAVE F4
CAM  7,PTEMP8+8  SET TEMP STORE STARTING ADDRESS
TNR  JB3  RESET OV, Z FLAGS
ADM  2,4096  OV NOT SET
JB3  ADM  JB4  SET OV FLAG
TJ  2,1  Z NOT ON
ADM  2,1  SET Z FLAG
JB4  ATN  7,1  SAVE FD=0
SFR  7,1  SAVE EXPONENT
SEX  M3  SAVE Z,OV FLAGS
ATN  7,1  AND EXPONENT
SFR  4  SAVE AMOST
SAM  7,1  TOPS 307  TOPS 308  TOPS 309  TOPS 310  TOPS 311  TOPS 312  TOPS 313  TOPS 314  TOPS 315  TOPS 316  TOPS 317  TOPS 318  TOPS 319  TOPS 320  TOPS 321  TOPS 322  TOPS 323  TOPS 324  TOPS 325  TOPS 326  TOPS 327  TOPS 328  TOPS 329  TOPS 330  TOPS 331  TOPS 332  TOPS 333  TOPS 334  TOPS 335  TOPS 336  TOPS 337  TOPS 338  TOPS 339  TOPS 340
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAL 7,1,</td>
<td>SAVE ALEAST</td>
<td>TOPS 341</td>
</tr>
<tr>
<td>CAD F1,</td>
<td>SAVE F1=IN</td>
<td>TOPS 342</td>
</tr>
<tr>
<td>SAM M7</td>
<td></td>
<td>TOPS 343</td>
</tr>
<tr>
<td>SFR 4,PTEMP8+7</td>
<td></td>
<td>TOPS 344</td>
</tr>
<tr>
<td>LFR 4,PTEMP8+14</td>
<td></td>
<td>TOPS 345</td>
</tr>
<tr>
<td>ADM 0,1</td>
<td>JUMP IF ENTERED</td>
<td>TOPS 346</td>
</tr>
<tr>
<td>JPM 0,REL14</td>
<td>BY PTRDWA</td>
<td>TOPS 347</td>
</tr>
<tr>
<td>ADM 0,1</td>
<td>CHECK FOR</td>
<td>TOPS 348</td>
</tr>
<tr>
<td>JZM 0,REL9B</td>
<td>PTRDWA ENTRY</td>
<td>TOPS 349</td>
</tr>
<tr>
<td>ADM 0,1</td>
<td>CHECK FOR</td>
<td>TOPS 350</td>
</tr>
<tr>
<td>JZM 0,REL9A</td>
<td>PTRDWA ENTRY</td>
<td>TOPS 351</td>
</tr>
<tr>
<td>ADM 0,1</td>
<td>RESET FOR CORRECT FORMAT</td>
<td>TOPS 352</td>
</tr>
<tr>
<td>CAD M1</td>
<td>LOAD WORD FOR PRINTING</td>
<td>TOPS 353</td>
</tr>
<tr>
<td>REL9B CAM 8</td>
<td>SET FOR NO PRECEEDING CHARACTER</td>
<td>TOPS 354</td>
</tr>
<tr>
<td>TRA REL11B</td>
<td>JUMP FOR PTRDWA ENTRY</td>
<td>TOPS 355</td>
</tr>
<tr>
<td>REL9A LFR 5,M1</td>
<td>LOAD PARAMETER WORD</td>
<td>TOPS 356</td>
</tr>
<tr>
<td>SBM 0,1</td>
<td>RESET CORRECT FORMAT</td>
<td>TOPS 357</td>
</tr>
<tr>
<td>CSM 5,M5</td>
<td>SET COUNTER</td>
<td>TOPS 358</td>
</tr>
<tr>
<td>CSM 6,1</td>
<td>SET QUAD. =-1</td>
<td>TOPS 359</td>
</tr>
<tr>
<td>REL9 CAD 4,1,</td>
<td>LOAD CURRENT WORD</td>
<td>TOPS 360</td>
</tr>
<tr>
<td>CJU 6,REL11A</td>
<td>JUMP ON WORD ON LINE COUNT</td>
<td>TOPS 361</td>
</tr>
<tr>
<td>CSM 6,6</td>
<td>SET WORD ON LINE COUNT</td>
<td>TOPS 362</td>
</tr>
<tr>
<td>REL11 SFN 4</td>
<td>SET S=-1</td>
<td>TOPS 363</td>
</tr>
<tr>
<td>REL11A CSM 8,3</td>
<td>SET S=3</td>
<td>TOPS 364</td>
</tr>
<tr>
<td>REL11B CSM 9,13</td>
<td>SET N=13</td>
<td>TOPS 365</td>
</tr>
<tr>
<td>CAM 10,13</td>
<td>SET K=13</td>
<td>TOPS 366</td>
</tr>
<tr>
<td>SBM 0,1</td>
<td>RESET CORRECT FORMULA</td>
<td>TOPS 367</td>
</tr>
<tr>
<td>CAM 13,10</td>
<td>SET FOR POSITIVE SIGN '+'</td>
<td>TOPS 368</td>
</tr>
<tr>
<td>CAM 2</td>
<td>SUPPRESS LEADING ZEROS</td>
<td>TOPS 369</td>
</tr>
<tr>
<td>TRA REL15</td>
<td>SKIP OVER</td>
<td>TOPS 370</td>
</tr>
<tr>
<td>REL14 CSM 8,M8</td>
<td>RESET ENTRY</td>
<td>TOPS 371</td>
</tr>
<tr>
<td>CSM 9,M9</td>
<td></td>
<td>TOPS 372</td>
</tr>
<tr>
<td>REL15 CAM 7</td>
<td>CLEAR SIGN FLAG</td>
<td>TOPS 373</td>
</tr>
<tr>
<td>SFR 2,PTEMP8+4</td>
<td>SAVE F2</td>
<td>TOPS 374</td>
</tr>
</tbody>
</table>
TZP REL17
STN F3
CAM 13,11
REL17 STR F3
JNM 8,REL20
JZM 8,REL21
CAM 1,63
CALL PTA
TRA REL21
REL20 CAM 1,56
CALL PTA
CJU 8,REL20
REL21 JZM 0,REL65
CAM 11,M10+M9-1
CAD F3
TZ REL32
CAM 8,M9+1
CAD 1.
JZM 8,REL25
J1 MPY 10.
CJU 8,J1
REL25 STR F2
REL26 CAD F3
DAV F2
TZP REL29
CAD F3
MPY 10.
STR F3
SBM 10,1
TRA REL26
REL29 CAD F2
MPY 10.
STR F2
REL30 CAD F3
LOAD F3 IN AMOST
JUMP IF AMOST G.T.E. ZERO
SET NUMBER POSITIVE
SET FOR MINUS SIGN '-'
PUT NUMBER IN F3
JUMP IF S G.T.E. 1
JUMP IF S = 0
PRINT
LF/CR
PROCEED
PRINT
SPACES
JUMP FOR ENTRY BY PTFDW
SET M1=-(N-K+1)
LOAD NUMBER IN AMOST
JUMP IF NUMBER=0
SET S=1-N
LOAD 1.0 INTO AMOST
JUMP IF S=0
MPY BY 10.0
FORM 10**W-1)
STORE 10**{N-1}
LOAD F3 IN AMOST
ABS(AO)-10**{N-1}IN AMOST
ABS(AO)G.T.E. 10**{N-1}
FORM
10AO
SUBTRACT 1 FROM EXPONENT
FOR EVERY MULTIPLICATION
RETURN TO TRY AGAIN
10**(N-1)IN AMOST
10**N
LOAD F3 IN AMOST
TOPS 375
TOPS 376
TOPS 377
TOPS 378
TOPS 379
TOPS 380
TOPS 381
TOPS 382
TOPS 383
TOPS 384
TOPS 385
TOPS 386
TOPS 387
TOPS 388
TOPS 389
TOPS 390
TOPS 391
TOPS 392
TOPS 393
TOPS 394
TOPS 395
TOPS 396
TOPS 397
TOPS 398
TOPS 399
TOPS 400
TOPS 401
TOPS 402
TOPS 403
TOPS 404
TOPS 405
TOPS 406
TOPS 407
TOPS 408
CAM 1,M13
CALL PTA
REL44 JUM 11,REL45
CAM 1,41
CALL PTA
REL45 CJU 15,REL48
SBM 12,10
JUM 12,REL47
CSM 12,9
ADM 10,1
REL47 ADM 12,10
REL48 CAM 1,M12
CALL PTA
TRA REL50
REL49 LDM 3,PTEMP8+14
JZM 3,REL50
CAM 1,56
CALL PTA
REL50 SFN M12
CAD 0.
MPY F2
ADD F3
MPY 10.
STR F3
JNM 9,REL40
JZM 0,REL59
CAM 1,80
CALL PTA
CAD 10.
STR F2
ATN M10
CAD 0.
JPM 10,REL56
STN F3
CHARACTER
JUMP IF POINT ALL READY PRINTED
OUTPUT
DECIMAL POINT
JUMP EXCEPT IN FIRST PART
JUMP IF D.N.E. 10
IF D=10, SET D=1
AND INCREASE EXPONENT BY 1
PRINT
D
JUMP IF SUPPRESSED LEADING ZEROS
ARE REPLACED BY NOTHING
OUTPUT
SPACE
FORM
A=10\{(A-\{10^{(N-1)}\})D\}
PUT EXPONENT
INTO AMOST
JUMP IF EXPONENT POSITIVE
CHANGE SIGN
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATN 1</td>
<td>PRINT '-'</td>
</tr>
<tr>
<td>CAM 1,10</td>
<td>PRINT '+'</td>
</tr>
<tr>
<td>CALL PTA</td>
<td></td>
</tr>
<tr>
<td>STR F3</td>
<td></td>
</tr>
<tr>
<td>CAM 0</td>
<td>CHANGE FORMAT</td>
</tr>
<tr>
<td>CSM 9,2</td>
<td>SET N=2</td>
</tr>
<tr>
<td>CSM 11,3</td>
<td>-(N-K+1)=-3</td>
</tr>
<tr>
<td>TRA REL40</td>
<td></td>
</tr>
<tr>
<td>REL59</td>
<td></td>
</tr>
<tr>
<td>LFR 7,PTEMP8+5</td>
<td>RESTORE F4</td>
</tr>
<tr>
<td>LFR 2,PTEMP8+4</td>
<td>RESTORE F2</td>
</tr>
<tr>
<td>ADM 8,1</td>
<td>(M8)+1</td>
</tr>
<tr>
<td>JPM 8,REL62</td>
<td></td>
</tr>
<tr>
<td>CSM 0,1</td>
<td>RESET CORRECT FORMAT</td>
</tr>
<tr>
<td>CJS 5,REL9</td>
<td></td>
</tr>
<tr>
<td>REL62</td>
<td></td>
</tr>
<tr>
<td>LFR 3,PTEMP8+2</td>
<td>SET TEMP STORE STARTING ADDRESS</td>
</tr>
<tr>
<td>CAM 1,PTEMP8+8</td>
<td>RESTORE</td>
</tr>
<tr>
<td>CAD 1,1</td>
<td>F0=OUT</td>
</tr>
<tr>
<td>SAM F0</td>
<td></td>
</tr>
<tr>
<td>ATN 1,1</td>
<td>FETCH Z,OV FLAGS</td>
</tr>
<tr>
<td>LFR 5</td>
<td>AND EXPONENT</td>
</tr>
<tr>
<td>JPM 6,JB5</td>
<td>JUMP IF OV NOT SET</td>
</tr>
<tr>
<td>CAD 13</td>
<td>SET OV</td>
</tr>
<tr>
<td>DIV 15,3,2</td>
<td></td>
</tr>
<tr>
<td>JB5</td>
<td></td>
</tr>
<tr>
<td>CAD 1,1</td>
<td>LOAD AMOST</td>
</tr>
<tr>
<td>LAL 1,1</td>
<td>LOAD ALEAST</td>
</tr>
<tr>
<td>CAE M7</td>
<td>RESTORE EXPONENT</td>
</tr>
<tr>
<td>CRM 6,1</td>
<td></td>
</tr>
<tr>
<td>JPM 6,JB6</td>
<td>JUMP IF Z NOT SET</td>
</tr>
<tr>
<td>ADE -128</td>
<td>SET Z</td>
</tr>
<tr>
<td>ADE -128</td>
<td></td>
</tr>
<tr>
<td>JB6</td>
<td></td>
</tr>
<tr>
<td>ATN 1,1</td>
<td>RESTORE</td>
</tr>
<tr>
<td>LFR 5</td>
<td>F1=IN</td>
</tr>
<tr>
<td>OCTQ 10727</td>
<td></td>
</tr>
<tr>
<td>LFR 4,PTEMP8+1</td>
<td>RESTORE F4</td>
</tr>
</tbody>
</table>
LFR 5,PTMP8+3  RESTORE F5  TOPS 511
LFR 6,PTMP8  RESTORE F6  TOPS 512
JLH M3  RETURN  TOPS 513
REL65 CSM 8, M10  (M8) = -K  TOPS 514
CAD 1.  TOPS 515
JZM 8, REL67  TOPS 516
REL66 MPY 10.  TOPS 517
CJU 8, REL66  TOPS 518
REL67 MPY  F3  (10^N) A0  TOPS 519
STR F3  TOPS 520
REL68 CAM 8, M9  (M8) = - N  TOPS 521
CAD 1.  TOPS 522
J2 MPY 10.  TOPS 523
CJU 8, J2  TOPS 524
SUB F3  (10^N) - (10^N) A0  TOPS 525
TP REL71  TOPS 526
SBM 9, 1  N = N + 1  TOPS 527
TRA REL68  TOPS 528
REL71 CAM 11, M9 + M10 - 1  TOPS 529
TRA REL33  TOPS 530
FIL GO  TOPS 531
GO  TOPS 532

*** NUMBER 9) PUNCH OR TYPE DECIMAL WORD WITH ADDRESS

ENTRY PTDWA  TOPS 533
PTMP8A CALL PTDW2  NEEDED TO SET UP TRANSFER VECTOR  TOPS 534
PTDW2 CALL PTQ1  FREE F4  TOPS 535
PTQ1 CALL PTQ2  GO TO OUTPUT ADDRESS  TOPS 536
PTQ2 LFR 4, PTMP8A  TOPS 537
PTQ2 CALL  TOPS 538
TRA PTDW2  RESTORE F4  TOPS 539
PTQ2 GO TO OUTPUT WORD  TOPS 540
PTQ2
*** NUMBER 10) MEMORY DUMP CONTROL FOR FULL WORD OCTAL
ENTRY PTMDF
PTMP6A BSS 1
CALL PTFW1
PTMDF SFR 4,PTMP6A FREE F4
CAM ,PTMP6A+1 LOAD LOCATION OF JUMP INSTRUCTION
CALL PTMDX
GO
*** NUMBER 11) MEMORY DUMP CONTROL FOR FULL WORD OCTAL WITH ADDRESSES
ENTRY PTMDFA
PTMP6B BSS 1
CALL PTFWA
PTMDFA SFR 4,PTMP6B FREE F4
CAM ,PTMP6B+1 LOAD LOCATION OF JUMP INSTRUCTION
CALL PTMDX
GO
*** NUMBER 12) MEMORY DUMP CONTROL FOR SEXADECIMAL WORD
ENTRY PTMDS
PTMP6C BSS 1
CALL PTSW1
PTMDS SFR 4,PTMP6C FREE F4
CAM ,PTMP6C+1 LOAD LOCATION OF JUMP INSTRUCTION
CALL PTMDX
GO
*** NUMBER 13) MEMORY DUMP CONTROL FOR SEXADECIMAL WORD WITH ADDRESSES
ENTRY PTMDSA
PTMP6D BSS 1
CALL PTSWA
PTMDSA SFR 4,PTMP6D FREE F4
CAM ,PTMP6D+1 LOAD LOCATION OF JUMP INSTRUCTION
CALL PTMDX
GO
**NUMBER 14) MEMORY DUMP CONTROL FOR DECIMAL WORD**

**ENTRY** PTMDD

PTMP6E BSS 1  
CALL PTDWI

PTMDD SFR 4,PTMP6E  
CAM 5,PTMP6E+1  
CALL PTMDX  
GO

**FREE F4**

**LOAD LOCATION OF JUMP INSTRUCTION**

**NUMBER 15) MEMORY DUMP CONTROL FOR DECIMAL WORD WITH ADDRESSES**

**ENTRY** PTMDA

PTMP6F BSS 1  
CALL PTDWA

PTMDDA SFR 4,PTMP6F  
CAM 5,PTMP6F+1  
CALL PTMDX  
GO

**FREE F4**

**LOAD LOCATION OF JUMP INSTRUCTION**

**NUMBER 16) PUNCH OR TYPE MEMORY DUMP**

**ENTRY** PTMDX

PTEMP6 BSS 1

PTMDX SFR 5,PTEMP6  
LFR 5,MO  
SFR 5,PTMDX1  
LDM 2,M1  
LDM 1,M1  
CSM 2,M2-M1+1  
FIL

PTMDX1 JSB 3,PTMDX  
FIL

ADM 1,1  
CJU 2,PTMDX1  
LFR 5,PTEMP6  
LFR 4,MO-1  
JLH M3  
GO

**FREE F5**

**FETCH JUMP INSTRUCTION**

**STORE JUMP INSTRUCTION**

**FETCH L.W.A.**

**FETCH F.W.A.**

**SET WORD COUNTER**

**(JUMP TO PROPER SUBROUTINE)**

**(IS STORED HERE)**

**INCREMENT WORD COUNT**

**MORE WORDS TO BE DUMPED**

**RESTORE F5**

**RESTORE F4**

**EXIT**

**TOPS 569**  
**TOPS 570**  
**TOPS 571**  
**TOPS 572**  
**TOPS 573**  
**TOPS 574**  
**TOPS 575**  
**TOPS 576**  
**TOPS 577**  
**TOPS 578**  
**TOPS 579**  
**TOPS 580**  
**TOPS 581**  
**TOPS 582**  
**TOPS 583**  
**TOPS 584**  
**TOPS 585**  
**TOPS 586**  
**TOPS 587**  
**TOPS 588**  
**TOPS 589**  
**TOPS 590**  
**TOPS 591**  
**TOPS 592**  
**TOPS 593**  
**TOPS 594**  
**TOPS 595**  
**TOPS 596**  
**TOPS 597**  
**TOPS 598**  
**TOPS 599**
*** NUMBER 17) FAST REGISTER DUMP CONTROL FOR FULL WORD OCTAL
ENTRY PTFRD0
PTFRDO CAM 2,PTMP7A LOAD LOCATION OF JUMP INSTRUCTION
CAM 0,M3 SAVE LINK
CALL PTFRD
PTMP7A CALL PTFW2
GO

*** NUMBER 18) FAST REGISTER DUMP CONTROL FOR SEXADECIMAL WORD
ENTRY PTFRDS
PTFRDS CAM 2,PTMP7B LOAD LOCATION OF JUMP INSTRUCTION
CAM 0,M3 SAVE LINK
CALL PTFRD
PTMP7B CALL PTSW2
GO

*** NUMBER 19) FAST REGISTER DUMP CONTROL FOR DECIMAL WORD
ENTRY PTFRDD
PTFRDD CAM 2,PTMP7C LOAD LOCATION OF JUMP INSTRUCTION
CAM 0,M3 SAVE LINK
CALL PTFRD
PTMP7C CALL PTDW2
GO

*** NUMBER 20) PUNCH OR TYPE FAST REGISTER DUMP
ENTRY PTFRD
PTMP7 BSS 7
PTFRD SFR 7,PTMP7+5 FREE F7
LFR 7,M2 FETCH JUMP INSTRUCTION
SFR 7,PTFRDB STORE JUMP INSTRUCTION
LFR 7,M1 FETCH ORIGINAL F4
SFR 7,PTMP7+2 STORE ORIGINAL F4
<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM 12,2948</td>
<td>12,2948 SET LINK</td>
<td>O(2948 = 05604) SET LINK</td>
<td>TOPS 625</td>
</tr>
<tr>
<td>CAM 13,00</td>
<td>13,00</td>
<td></td>
<td>TOPS 626</td>
</tr>
<tr>
<td>SFR 7,PTFRD2</td>
<td>7,PTFRD2 STORE LINK</td>
<td>STORE LINK</td>
<td>TOPS 627</td>
</tr>
<tr>
<td>SFR 2,PTEMP7</td>
<td>2,PTEMP7 STORE OTHER FAST Registers</td>
<td>STORE OTHER FAST REGISTERS</td>
<td>TOPS 628</td>
</tr>
<tr>
<td>SFR 3,PTEMP7+1</td>
<td>3,PTEMP7+1</td>
<td>SET UP SPECIAL CHARACTER</td>
<td>TOPS 629</td>
</tr>
<tr>
<td>SFR 5,PTEMP7+3</td>
<td>5,PTEMP7+3</td>
<td>SET UP 'LF/CR'</td>
<td>TOPS 630</td>
</tr>
<tr>
<td>SFR 6,PTEMP7+4</td>
<td>6,PTEMP7+4</td>
<td>SET UP 'F'</td>
<td>TOPS 631</td>
</tr>
<tr>
<td>CAM 1,48</td>
<td>1,48</td>
<td>SET UP SPECIAL CHARACTER</td>
<td>TOPS 632</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>0,6</td>
<td>GO TO OUTPUT CHARACTER</td>
<td>TOPS 633</td>
</tr>
<tr>
<td>CSM 0,6</td>
<td>0,6</td>
<td>SET LOOP COUNTER</td>
<td>TOPS 634</td>
</tr>
<tr>
<td>CAM 1,63</td>
<td>1,63</td>
<td>SET UP 'LF/CR'</td>
<td>TOPS 635</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>1,81</td>
<td>GO TO OUTPUT IT</td>
<td>TOPS 636</td>
</tr>
<tr>
<td>CAM 1,81</td>
<td>1,81</td>
<td>SET UP 'F'</td>
<td>TOPS 637</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>1,91</td>
<td>GO TO OUTPUT IT</td>
<td>TOPS 638</td>
</tr>
<tr>
<td>CAM 1,91</td>
<td>1,91</td>
<td>SET UP NUMBER OF FAST REGISTER</td>
<td>TOPS 639</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>1,91</td>
<td>GO TO OUTPUT IT</td>
<td>TOPS 640</td>
</tr>
<tr>
<td>CAM 1,91</td>
<td>1,91</td>
<td>SET ADDRESS OF FAST REGISTER</td>
<td>TOPS 641</td>
</tr>
<tr>
<td>PTFRDA CAM 1,PTA</td>
<td>1,PTA</td>
<td>SET UP 'LF/CR'</td>
<td>TOPS 642</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>1,PTA</td>
<td>GO TO OUTPUT IT</td>
<td>TOPS 643</td>
</tr>
<tr>
<td>CAM 1,PTA</td>
<td>1,PTA</td>
<td>SET UP NUMBER OF FAST REGISTER</td>
<td>TOPS 644</td>
</tr>
<tr>
<td>CALL PTA</td>
<td>1,PTA</td>
<td>GO TO OUTPUT IT</td>
<td>TOPS 645</td>
</tr>
<tr>
<td>CAM 1,PTA</td>
<td>1,PTA</td>
<td>SET ADDRESS OF FAST REGISTER</td>
<td>TOPS 646</td>
</tr>
<tr>
<td>PTFRDA</td>
<td>3,PTFRD</td>
<td>(JUMP TO PROPER SUBROUTINE IS STORED HERE)</td>
<td>TOPS 647</td>
</tr>
<tr>
<td>FIL</td>
<td>0,PTFRDA</td>
<td>MORE TO GO</td>
<td>TOPS 648</td>
</tr>
<tr>
<td>LFR 4,PTEMP7+2</td>
<td>4,PTEMP7+2</td>
<td>RESTORE F4</td>
<td>TOPS 649</td>
</tr>
<tr>
<td>LFR 7,PTEMP7+5</td>
<td>7,PTEMP7+5</td>
<td>RESTORE</td>
<td>TOPS 650</td>
</tr>
<tr>
<td>FIL</td>
<td>0,PTFRDA</td>
<td>MORE TO GO</td>
<td>TOPS 651</td>
</tr>
<tr>
<td>PTFRD2</td>
<td>M3</td>
<td>EXIT (LINK IS STORED HERE)</td>
<td></td>
</tr>
<tr>
<td>ENTRY</td>
<td>PTACC</td>
<td>TOPS 652</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>PTEMS5</td>
<td>BSS</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>63, 76, 78, 78, 96, 88, 96, 87, 76, 95, 90, 93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>56, 79, 96, 88, 91, 63, 76, 24, 26, 30, 31, 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>56, 56, 8191, 56, 56, 56, 90, 97, 56, 20, 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>15, 20, 14, 12, 31, 26, 29, 56, 26, 25, 56, 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTAMS2</td>
<td>DECEQ</td>
<td>63, 76, 23, 16, 12, 30, 31, 56, 56, 8191,</td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>56, 56, 56, 101, 56, 20, 25, 15, 20, 14, 12, 31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>26, 29, 56, 26, 25, 56, 56, 56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTAMS3</td>
<td>DECEQ</td>
<td>63, 93, 105, 80, 94, 56, 56, 56, 56, 8191,</td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>56, 56, 56, 90, 97, 56, 30, 16, 31, 56, 15, 32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECEQ</td>
<td>29, 20, 25, 18, 56, 30, 31, 26, 29, 12, 18, 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTAMS4</td>
<td>DECEQ</td>
<td>63, 81, 11, 90, 32, 31, 56, 56, 8191,</td>
<td></td>
</tr>
<tr>
<td>PTAMS5</td>
<td>DECEQ</td>
<td>63, 81, 11, 84, 25, 56, 56, 8191</td>
<td></td>
</tr>
<tr>
<td>PTACC</td>
<td>SFR</td>
<td>4, PTEMS5+2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SFR</td>
<td>5, PTEMS5+3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SFR</td>
<td>6, PTEMS5+4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SFR</td>
<td>7, PTEMS5+5</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>8</td>
<td>SET FLAGS TO ZERO</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>0, PTEMS5+6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNOPTACC1</td>
<td>SAVE F0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TNOPTACC2</td>
<td>SET BASE ADDRESS OF STORAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADNM</td>
<td>11, 1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADNM</td>
<td>9, 1</td>
<td>SET FLAG- OV ON</td>
<td></td>
</tr>
<tr>
<td>PTACC2</td>
<td>SAM: 11,1,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAL: 11,1,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEX: 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TU: PTACC3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM: 10,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTACC3</td>
<td>CAD: F1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAM: M11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SFR: 7,PTEMP5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 1,PTAMS1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL: PTMSS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFR: 5,PTEMP5+7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 1,M15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL: PTWS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 1,PTAMS2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JZM: 8,PTACC4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBM: 1,5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAD: 13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV: 15,3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTACC4</td>
<td>CALL: PTMSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFR: 5,PTEMP5+8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 1,M15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CALL: PTWS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 1,PTAMS3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JZM: 10,PTACC5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBM: 1,5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTACC5</td>
<td>CALL: PTMSS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAD: M11-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAM: F5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SEX: 15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAM: 0,PTAMS4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUM: 9,PTACC7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUM: 10,PTACC6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TU: PTACC8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- SAVE A-MOST
- SAVE A-LEAST
- SAVE EXPONENT
- SET FLAG-Z ON
- SAVE F1
- SAVE EXPONENT
- LOAD FIRST MESSAGE
- OUTPUT FIRST MESSAGE
- OUTPUT A-MOST
- LOAD SECOND MESSAGE
- 'OV ON' FLAG NOT SET
- ADJUST SECOND MESSAGE
- RESET 'OV'
- OUTPUT SECOND MESSAGE
- OUTPUT A-LEAST
- LOAD THIRD MESSAGE
- 'Z ON' FLAG NOT SET
- ADJUST THIRD MESSAGE
- OUTPUT THIRD MESSAGE
- FETCH R,ES
- LOAD FOURTH MESSAGE
- **'OV ON AFTER R,ES'' FLAG SET
- 'Z ON' FLAG SET

Page: 38 of 39
PTACC6  CALL PTWS
PTACC6  CAM  1,M15
PTACC7  ADM  15,128
PTACC7  SBM  0,6
PTACC8  CAM  1,M15
PTACC9  CAM  1,M0
PTACC9  CALL PTWS
PTACC9  CALL PTMSS
PTACC9  CAM  1,PTEMP5+1
PTACC9  CALL PTSM
PTACC9  CAM  1,PTAMS5
PTACC9  CALL PTMSS
PTACC9  CAM  1,PTEMP5+9
PTACC9  CALL PTSM
PTACC9  CAD  PTEMP5+1
PTACC9  SAM  PTEMP5+1
PTACC9  CAD  PTEMP5+7
PTACC9  LAL  PTEMP5+8
PTACC9  LFR  7,PTEMP5
PTACC9  CAE  M15
PTACC9  JZM  10,PTAC10
PTACC9  ADE  -128
PTACC9  ADE  -128
PTAC10  LFR  7,PTEMP5+9
PTAC10  DGTQ  10737
PTAC10  LFR  5,PTEMP5+3
PTAC10  LFR  6,PTEMP5+4
PTAC10  LFR  7,PTEMP5+5
PTAC10  LFR  4,PTEMP5+2
PTAC10  JIH  M3
PTAC10  GO

OUTPUT R,ES
SET UP QUESTION MARK
OUTPUT QUESTION MARK
CORRECT ES
ADJUST FOURTH MESSAGE
OUTPUT R,ES
LOAD PARAMETER
OUTPUT FOURTH MESSAGE
OUTPUT F0 IN SEXADECIMAL
LOAD FIFTH MESSAGE
OUTPUT FIFTH MESSAGE
OUTPUT F1 IN SEXADECIMAL
RELOAD F0
RELOAD A-MOST
RELOAD A-LEAST
RELOAD EXPONENT
ADJUST
FOR
UNDERFLOW
RELOAD F1
EXIT
NAME: I/O List Program

PURPOSE: This program provides the tape blocking and buffering necessary for the FORTRAN statements

READ TAPE
WRITE TAPE
BACKSPACE
REWIND

and

END FILE

It can be used by other programs, but, due to the nature of the FORTRAN statements, it is not an efficient way of reading from tapes.

TEMPORARY STORAGE: Accumulator, FO-F3 and the first 256 words of COMMON.

NUMBER OF WORDS: 76 words

USE: The program uses a calling sequence identical to that of the PRINT/READ/PUNCH program (so that the FORTRAN compiler task is identical).

A CALL is made with the address of an I/O list in ML. This list defines the operation and a number of groups of contiguous memory cells in the following way:

| C | M | N | T |

Format of the I/O List Word
USE (Continued): The first quarter of the list is the control word. It is split into 2 and 11 bit parts.

\[
\begin{array}{ll}
\hline
2 & 11 \\
\hline
\end{array}
\]

Control Quarter

The 11-bit group indicates the function to be performed:

\[
\begin{array}{ll}
0 & \text{Read} \\
1000 & \text{Write} \\
2040 & \text{Rewind} \\
2050 & \text{Backspace} \\
2120 & \text{End of File} \\
\end{array}
\]

The last three are "control operations." In these cases the list is one word long and the two-bit group is ignored. For all operations, T is the logical tape unit number.

For read and write operations, M is the first word of a group of length N (≤ 4095) words which are to be transmitted. If the first of the top two bits of the control quarter is a one, then another list word follows in the next location in core. T and the 11 operation bits are ignored in all subsequent words of the list.

When the top bit of a control quarter is a zero, the list terminates. The CALL on the I/P LIST program is said to be a partial or final CALL accordingly as the second bit of the control quarter is a one or a zero.

A partial CALL means that the next CALL on the I/P LIST program is for the same unit and with the same control. The new list is used to add more words to the block of words specified by the preceding list.
USE (Continued): A final CALL causes the block of words to be output to tape (WRITE), or it causes the program to be set so that the next READ will start a fresh block from tape.

METHOD: Information put on a tape by FORTRAN object program must be buffered in records of not more than 256 words. Since a block of data may be longer than this, one block may occupy several records. To accomplish this the first word of each record is a control word with the format:

```
  [record number with block] [256 + number of words or 0 if there is another block]
```

The words are placed into a buffer backwards until it is full (255 words) and then another record is started. When the last word of a block has been placed in the buffer, the control word is placed in the next available position and a record of \( n + 1 \) words is written where \( n \) is the number of data words in the buffer.

READING tape is performed by loading the buffer with the first block and then copying as many words as desired into the cells defined in the list. If the buffer is exhausted it is refilled from tape as long as there are records available. If the tape block runs out first, a message is printed, and an exit occurs to SYSERR. Otherwise the tape is moved to the correct end of the block.

BACKSPACE is a slow operation since the tape is backspaced one block, read for one block and then backspaced the number of blocks specified in the record number quarter of the last block.
METHOD (Continued): A control word is used to tell whether the buffer is free or not, and if not, which unit is using it.

\[
\begin{array}{c|c|c}
00400 & 0037 & - \\
\end{array}
\]

- Number of records in this block
- Unit using buffer, 0 => free
- End of buffer address

BUF+1: BUFFER CONTROL WORD

A control word is also used to remember whether the last CALL was partial or final.

\[
\begin{array}{cccc}
& & & \\
\end{array}
\]

- 0 if FINAL CALL,
- Previous Control
- Counters saved from last entry if it was a partial CALL
- (partial CALL control)

BUF

Use of Modifiers.

M0 is 0 if the program is input mode.

M1 contains the next list address word.

M4-M7 contains the current list word, the address in M5 is incremented and the count in M6 is decremented.

M8 is positive during read if there are no more records in this block.

M9 contains the buffer address as the buffer is loaded or unloaded.

M10 contains a count of 255 during write or a count of the number of words in a record during READ. It determines when a block is full or empty.

M11 contains the unit number currently in use.
METHOD (Continued): After a partial call, the information in F6 is stored in BUF and restored at the next CALL.
IO LIST
SUBROUTINE FLOW
ENTRY       IOLIST
READ       EQU  0
WRITE      EQU  512
BCKSPR     EQU  1084
WAIT       EQU  256
IOLIST     SFR  5,T2
           SFR  6,T3
           SFR  7,T4
           SFR  4,T1
           LFR  6,BUF
           ATN  1,1
           LFR  5
           JUM  8,IOL18
           CAM  8,M4
           ORM  8,4096
           CAM  11,M7
           LFR  7,BUF+1
           CAM  13
           SFR  7,BUF+1
           CRN  4,11
           CAM  12
           JPM  12,IOL4
           ANN  4,2047
           CAM  12
           CAM  15,M7+1024
           SFR  7,IOL2
           CALL  SYSAUX
IOL2       BSS  1
           SBM  12,BCKSPR
           JUM  12,IOL20
           CAM  4,1024
           CAM  6,1

SAVE F4-F7
LOAD BUF
LOAD FIRST CONTROL WORD
SECONDARY ENTRY JUMP
SET CONTROL AND
UNIT IN BUF,0 AND BUF,3
RECORD COUNT = 0
NOT CONTROL
CONTROL CODE
TAPE UNIT NUMBER
NOT BACKSPACE, EXIT
<table>
<thead>
<tr>
<th>IOL 4</th>
<th>ANN</th>
<th>8,512</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>10,-1</td>
<td></td>
</tr>
<tr>
<td>JZM</td>
<td>0,IOL6A</td>
<td></td>
</tr>
<tr>
<td>IOL 6</td>
<td>LFR</td>
<td>10,-256</td>
</tr>
<tr>
<td></td>
<td>JZM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SFR</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>CALL</td>
<td>SYSAUX</td>
<td></td>
</tr>
<tr>
<td>IOL 3</td>
<td>BSS</td>
<td>1</td>
</tr>
<tr>
<td>ADM</td>
<td>13,1</td>
<td></td>
</tr>
<tr>
<td>SFR</td>
<td>7, BUF+1</td>
<td></td>
</tr>
<tr>
<td>CAM</td>
<td>9, M14</td>
<td></td>
</tr>
<tr>
<td>IOL 6</td>
<td>ADM</td>
<td>6,-1</td>
</tr>
<tr>
<td>JNM</td>
<td>6, IOL8</td>
<td></td>
</tr>
<tr>
<td>CJZ</td>
<td>10, IOL10A</td>
<td></td>
</tr>
<tr>
<td>JZM</td>
<td>0, IOL7</td>
<td></td>
</tr>
<tr>
<td>CAD</td>
<td>5,1,9</td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>M9</td>
<td></td>
</tr>
<tr>
<td>IOL 5</td>
<td>SBM</td>
<td>9,1</td>
</tr>
<tr>
<td>TRA</td>
<td>IOL6</td>
<td></td>
</tr>
<tr>
<td>FIL</td>
<td>IOL7</td>
<td></td>
</tr>
<tr>
<td>CAD</td>
<td>M9</td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>5,1,9</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>IOL5</td>
<td></td>
</tr>
<tr>
<td>IOL 8</td>
<td>JPM</td>
<td>4, IOL9</td>
</tr>
<tr>
<td></td>
<td>ATN</td>
<td>1,1,9</td>
</tr>
<tr>
<td></td>
<td>LFR</td>
<td>5</td>
</tr>
<tr>
<td>TRA</td>
<td>IOL6</td>
<td></td>
</tr>
<tr>
<td>IOL 9</td>
<td>CRN</td>
<td>4,12</td>
</tr>
<tr>
<td>CAM</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>JPM</td>
<td>12, IOL10B</td>
<td></td>
</tr>
<tr>
<td>ORM</td>
<td>8,2048</td>
<td></td>
</tr>
<tr>
<td>IOL 12</td>
<td>SFR</td>
<td>6, BUF</td>
</tr>
</tbody>
</table>

MO=O MEANS INPUT
BLOCK COUNT EMPTY FOR INPUT
INPUT

BLOCK EMPTY FOR OUTPUT
BUFFER FREE

WAIT FOR UNIT NOT BUSY
INCREASE RECORD COUNT

WORD COUNT DECREASED
TEST FOR EMPTY
BUFFER EXHAUSTED
TEST INPUT

MOVE TO BUFFER
BUFFER COUNT

MOVE FROM BUFFER
JUMP IF LAST CONTROL
NEXT CONTROL TO F5

JUMP IF FINAL CONTROL
RESTORE BUF
LFR 4, T1
LFR 5, T2
LFR 6, T3
LFR 7, T4
JLH M3
IOL 10B JUM 0, IOL 10
JPM 8, IOL 20
IOL 14 JPM 8, IOL ERR
LFR 7, IOL 15
CAM 15, M11 + 1024
SFR 7, IOL 15
CALL SYSAUX
IOL 15 DECQ READ, RDBF-255, IOLEOF, 0
CAM 12, WAIT
SFR 7, IOL 16
CALL SYSAUX
IOL 16 BSS 1
CRN 4, 11
CAM 12
JNM 12, IOL 21
CAM 9, RDBF-255
LFR 7, M9
ORM 8, 4096
JZM 14, IOL 17
ANM 8, 4095
ADM 14, 1
IOL 17 ADM 6, 1
ADM 9, M14 + 255
CAM 10, -256-M14
TRA IOL 6
IOL 10A JZM 0, IOL 14
IOL 10 LFR 7, BUF + 1
CAM 14, M10
SFR 7, M9
CAM 13, M9

AND F4-F7
JUMP IF OUTPUT
NO BLOCKS LEFT JUMP
NO BLOCKS LEFT - ERROR
READ TAPE
WAIT UNTIL IN
BACKSPACE
MORE BLOCKS TO COME
RESTORE WORD COUNT
END OF BUFFER SET COUNT
JUMP IF INPUT
BUFFER COUNT STORE CONTROL
CAM  12, WRITE
CAM  15, M11+1024
SFR  7, IOL11
CALL SYSAUX

IOL11 BSS  1
LFR  7, BUF+1
CAM  15, M11+1024
SFR  7, BUF+1
ADM  6, 1
JZM  10, IOL13

IOL20 CAM  8
TRA  IOL12

IOL18 ANN  8, 512
CAM  0
TRA  IOL6

IOL21 LFR  7, RDBF=255
CSM  11, M13
LFR  7, IOL22
CAM  15, M7+1024
SFR  7, IOL22

IOL23 CALL SYSAUX
IOL22 DEQ  BCKSPR, 0, 0, 0
CJU  11, IOL23
TRA  IOL20

BUF BSS  1
DEQ  WAIT, 0, RDBF, 0
ASSIGN T1, T2, T3, T4
RDBF EQU  255

IOLERR CALL SYSIO
DEQ  WRITE+2, MESS, 0, 0

MESS CHR  32, 6 READ TAPE LIST TOO LONG.
NAME:                  Compare N Words
PURPOSE:              Subroutine to compare two lists of N words
NUMBER OF WORDS:      10
TEMPORARY STORAGE:     3 words in COMMON
FAST REGISTERS CHANGED: None if two lists are identical; F4 if two lists are not identical.
EXECUTION TIME:       Variable, depending on the parameter N
USE:                  This subroutine is useful mainly for engineering purposes. The parameters are specified as follows:

          CAM  1,PARAM
          CALL  CMPL

where PARAM is the address of a full word in memory containing the following parameters:

Quarter word 0           N           N words in the two lists
Quarter word 1           FWA1        First word address of list 1
Quarter word 2           FWA2        First word address of list 2
Quarter word 3           ERRTRN      Transfer address if compare error

N lies in the range

1 ≤ N ≤ 8192
ERRTRN must be the address of the first quarter of a word in memory as control is transferred by a JLH M3 order. Where an error is detected and control is returned to the main program via ERRTRN, $M_1$ and $M_2$ will contain the addresses of the two words in question plus 1. For example, if the word in location 300 does not agree with the word in location 500, a return is made via ERRTRN with $M_1 = 301$ and $M_2 = 501$. 
ENTRY CMPI CMP1

CMPI SFR 4, COMMON
SFR 6, COMMON+1
SFR 7, COMMON+2
LFR 4, M1
CSM 0, M0
CMPIA ATN 1, 1
LFR 6
ATN 2, 1
LFR 7
EOM 8, M12
EOM 9, M13
ORM 8
EOM 10, M14
ORM 8
EOM 11, M15
ORM 8
JUM 8, CMPIB
CMPIA 0, CMPIA
LFR 4, COMMON
CMPIB LFR 6, COMMON+1
LFR 7, COMMON+2
JLH M3
FIL
COMMON BSS 3

SAVE F4
SAVE F6
SAVE F7
LOAD PARAMETERS
SET COUNT FOR N WORDS
LOAD FIRST WORD
TO COMPARE
LOAD SECOND WORD
TO COMPARE
EOM M12 TO M8
EOM M13 TO M9
ORM M9 TO M8
EOM M14 TO M10
ORM M10 TO M8
EOM M15 TO M11
ORM M11 TO M8
JUMP IF NO COMPARE EQUAL
RETURN FOR MORE COMPARES
RESTORE F4
RESTORE F6
RESTORE F7
RETURN
TEMPORARY STORAGE

CMPI 000
CMPI 001
CMPI 002
CMPI 003
CMPI 004
CMPI 005
CMPI 006
CMPI 007
CMPI 008
CMPI 009
CMPI 010
CMPI 011
CMPI 012
CMPI 013
CMPI 014
CMPI 015
CMPI 016
CMPI 017
CMPI 018
CMPI 019
CMPI 020
CMPI 021
CMPI 022
CMPI 023
CMPI 024
NAME: PRINT, READ and PUNCH with input output conversion.

TEMPORARY STORAGE: None

NUMBER OF WORDS: 568 words

FAST REGISTERS CHANGED: Accumulator, F0 to F3, and M1 and M3 by the calling sequence.

USE: See Manual, Chapter 5.

DESCRIPTION: Master Control Logic-Main Format Scan Control

This portion of the program reads the format characters, sets the proper control words and switches and transfers to the parts of the program indicated by the format characters. READ, PUNCH or PRINT are each entrances to the master control logic.

If this is the first time READ, PUNCH or PRINT has been entered after a "final call," control words and the switches in M15 are set (see Fig. 2 for the meaning of the M15 switches; also see Figs. 4 and 5). If this is not the first time this routine has been entered a new I/O list word is put in F5, see Fig. 1. Parenthesis counts are initialized. Up to three nested parenthesis are allowed.

After initialization, the FORMRD routine is called and each format character is read, see Fig. 3, for the coding of each format character. Numbers, whose meaning is not yet defined, are stored in M9 (temporary counter).
DESCRIPTION (Continued): until a later format character defines their meaning; then they are placed in the proper modifier. See Fig. 1 for the complete fast register layout in the master logic. The following is a brief description of each portion of the program that is branched to when the proper format character is encountered. Blanks in the format string are ignored. The switches are kept in the low-order bits of ML5. They are interrogated by doing a CRNL5, bit position + 1, putting the result into ML4 and checking the sign of ML4. ML4 < 0 ⇒ on.

* → BSTAR: If this occurs inside a parenthesis, the program exits to a format error. If there is still more data in the I/O list to be processed, the program resets the format string to the last outside left parenthesis if there is one, or to the beginning of the format string if there are no parentheses. If there is no more data at this time, a repeat entry bit is examined to determine if a later portion of the program will produce more data to be output on this same line, if this is an output type. If there is, all of the counts are saved and the routine exits to the main program. If no repeat entry bit is on in an output type, the routine calls FRESET to fill out the rest of the line with blanks, prints or punches out the line and exits to the main program. For an input type, the routine just exits to the main program.

) → BRPR: It is first determined whether more repetitions are needed; if so the routine starts reading format at the previous matching left parenthesis. If no more repetitions are required at this level the routine pushes up to the next higher parenthesis level unless this parenthesis was one too many in which case it takes an error exit. The routine then starts processing data at this higher parenthesis level.
DESCRIPTION (Continued): \( \rightarrow \text{BSLH}: \) FRESET is called to finish this line and
the next format character is read.

\( \rightarrow \text{comma}: \) This will have been encountered while setting up previous format instruction and will have terminated it. Next character is now read.

\( \_ - 10 \rightarrow \text{BSUB2 and BSUB}: \) BSUB2 and BSUB are two entrances to a subroutine of the master logic which constructs a number from a decimally-represented integer until it reaches a character which terminates the construction. This routine is sometimes entered automatically when the previous format character implies that the next character should be a digit. Since an integer can be replaced by the letter N, this must be checked for first. If N is encountered, the integer portion of the current word in the I/O data block is used to fill out the format specification. Since zero has a BCD code of 10, this must be treated as a special case. If N was encountered, a short subroutine BINC is now called to move the I/O data list word address to the next full word boundary. BSUB2 (BSUB) now returns to the return address in control logic previously placed in M0.

(If a plus or minus had been encountered immediately before a digit and no other format character was associated previously with this digit, the routine now assumes that this integer is to be used for scaling and proceeds accordingly.)

\( S \rightarrow \text{BS}: \) This character indicates blanks are to be input or output. BSUB, see above, is called to determine how many blanks are required. The character terminating
DESCRIPTION (Continued): the S format is saved. If this is an output type blanks are output from M8, FLDEF is used; otherwise FRDEF is used to read in blanks.

\section*{X → EX:} This is the same as an S-type format, except the count has already been read in. This routine is now set up as an S type and the S routine is used.

\section*{F, I, E → BEF:} This routine first sets switches for E or F type and whether leading plus signs are to be printed. Whether or not the number is to be double-precision is also determined. The integer up to the decimal point is now constructed using BSWUB; this will be the total field length of the number to be input or output. The number following the decimal point indicates the number of decimal digits to be input or output. The scaling factor has already been set. If this is an output type, FDP is called as many times as required to print or punch the decimal numbers as indicated. If this is an input type, READEC is called to read in the decimal numbers indicated. The next format character is then read.

\section*{H → BH:} The number of hollerith characters involved has already been determined. If this is an output type, the characters immediately following the H in the format string are read by FORMRD, and output by FLDEF. If this is an input type, the characters following H in the format string are replaced by the characters in the input buffer. To get the proper location in the format string, the format control word, FRONT, see Fig. 2, is needed. If this routine had been entered from the A routine (see below), the routine returns to the A routine at BA3; otherwise the next format character is read. If this
DESCRIPTION (Continued): was an A or C format, the I/O list address and I/O count are incremented at BA3 since one A word may overlap into several list words. It is now determined whether there are more A characters in the current word. If there are, the next group of eight or less characters is processed as a continuation of the previous group. To do this the program branches to BA4 (see below under BA). If this word is finished the A field length is restored and BA4 gets the next data word. If there are more A words to be processed, the above is repeated as often as necessary. When finished the next format character is put in MO.

A or C → BA: BSUB is called to get the number of A characters desired. FRCNT, the format control word is saved because a fake format word will be constructed. When the A format is finished, program proceeds from where it left off in the format string given by FRCNT.

BA4 is the part of the BA routine which gets the data word to be processed by the BA routine. BINC is used to move up to the next full word boundary. If the I/O block is finished a new I/O block is gotten; otherwise the next I/O data word is gotten from the current block.

A fake Hollerith, H format, is constructed for eight or less characters. The A field length is decremented by eight, and a marker is set so that the H routine exits to BA3 (see above under BH). The current format word in the format string is replaced by the I/O data word (for output); otherwise by a blank word which is filled by BH from the input buffer. The BH routine is now used. When finished FRCNT and the current word in the format string are restored. If this is an input type, the A word constructed has been stored in the I/O data list.
DESCRIPTION (Continued): M → BM: Sets indicator for M, then uses BL.

D → BD: Sets indicator for D, then uses BL.

Q → BQ: Sets indicator for Q, then uses BL.

L → BL: BSUB is used to get the field length of the number. The next I/O data word is gotten. The following is what occurs for each case separately:

M input: Character is read from the input buffer using FRDF, blanks are ignored. The characters are assembled in M7 as octal characters. Note that since the BCD code for zero is 10, 0 must be treated differently. When a quarter word is assembled it is placed in the I/O list and the item count (number of words in block) is decreased. If there are more wanted, the above is repeated; otherwise routine exits to read the next format character.

M output: Puts the data block address plus item count into M7. The number is now printed in octal. Since the first digit of a five-digit octal number is binary, it must be converted differently from the second through fifth digits. The item comment is incremented and process is repeated as often as called for. Next format character is then read.

L input: This is identical to M input except that integers are read from the input buffer using READEC, the decimal read routine.

L output: This option is identical to M output except that the number is printed in decimal using FDP, the decimal print routine.

D input: Decimal number is read by READEC, and stored in proper quarter word of P7, which is then
stored in memory, counts are incremented and the process is repeated as often as specified. Next format character is then read.

D output: The data word specified by M5 is put in F7. The correct quarter is extracted for printing and put in the accumulator. It is then printed in decimal using FDP. The quarter word count and the item count is incremented. The above process is repeated as often as specified. Next format character is then read.

Q input: Identical to D input except that character is read from buffer using FRDF and constructed as an octal number.

Q output: Identical to D output except that the number is printed out octally. If more than five characters are printed, leading blanks are supplied. If less than five digits are required, the left most are suppressed. There is no zero suppression.

( → BLPR: BLPR is part of the initialization. The pushdown count is incremented unless the count exceeds three deep, in which case an error exit is taken. The address of this parenthesis is saved as is the number of repetitions for this parenthetical expression. The next character is then read.

P → BP: The integer scale factor has already been read. If scale factor is negative, this is indicated and it is stored as a positive number. The next character is then read.
DESCRIPTION (Continued): \( + \rightarrow \text{BPLUS} \) (\( - \) enters inside \( \text{BPLUS} \)): If a sign has already been encountered an error exit is taken. Otherwise plus or minus is indicated. The next format character is then read.

**FBACK**

If the format string is exhausted but data still remains to be processed, this subroutine returns to the format specification starting at the last outside left parenthesis, and if there is no parenthesis, it returns to the beginning of the format list. The remaining data is processed accordingly.

F4 and F7 are saved and the above format address is in M13. This specifies the format word to which we wish to return. This word is put into F7. The control word, FRCNT, Fig. 4, is put into F4, where the format counter will be reset to this new starting point. The proper format character is found and the running counts and switches are reset. The counts are saved and F4 and F7 are restored.

**FMRRD**

This subroutine reads the next format character. Format characters are packed two to a quarter word and the main purpose of this routine is to extract the proper quarter word and keep a running count of where we are in the format string. Counts are kept in FRCNT, cf., Fig. 4.

F4 and F7 are saved. F4 is loaded with FRCNT, and F7 with the current format word, specified in M3. All the information necessary to pick up the correct character.
DESCRIPTION (Continued): is kept in FRCNT. The character is extracted, put into M6, the counts are incremented, FRCNT is saved and F4 and F7 are restored.

**FLDBF**

FLDBF is the subroutine used to fill the output buffer character by character.

A buffer preparation word, FBFWD, is put in F7. This word is filled two characters to a quarter word, one character being added by each pass through this subroutine. When FBFWD is filled it is placed in its proper position in the output buffer.

F5 contains the control word, FCNTS, which keeps the running count of what is to be filled next, cf., Fig. 5. To put each character into FBFWD, the proper quarter word is first determined and then which half of the quarter word is to be filled. When FBFWD is filled it is placed in the output buffer, then restored to blanks, i.e., zeros.

If too many characters are to be put in the buffer, an error exit is taken.

After each pass, FBFWD and FCNTS are saved.

**FRDBF**

FRDBF is the subroutine used to read the next character from a card, the card already being in the input buffer.

The control word FCNTSC (cf., Fig. 6) with the running counts is put into F5. M5 is now used to pick up the correct word of the input buffer and put it into F7. If too many characters have been called for an error exit is taken.
DESCRIPTION (Continued): Since characters are packed two per quarter word, the correct quarter word and then the correct half of the quarter word is determined, the character extracted and put into M3. Counts are then incremented and FCNTSC is saved.

RESET

RESET is the subroutine used to output a line or card or read a card. The switches set in M15 determine which option is to be taken. SYSIØ is the subroutine used for input and output.

If input: SYSIØ is called to read a card into the input buffer INPBEF. The input control word FCNTSC is reset and saved, cf., Fig. 6.

If output: Output control word FCNTS, cf., Fig. 5, is put in F5, and the buffer preparation word FBFWD is put in F7. If the buffer is not completely filled already, it is filled out with blanks. If a line is to be printed, SYSIØ is called for printing. FCNTS is reset (-133 total character count) and FBFWD is zeroed. FCNTS and FBFWD are saved. Punching is the same, except that SYSIØ is called for punching, and the total character count is set to -80.

READEC

READEC is a subroutine which reads in decimal numbers and converts them to the proper octal representation. It makes use of two other subroutines--FRDBF, the read buffer subroutine which brings in characters one at a time from the input buffer, and FEXP, which provides

Date: 7/16/64
Section: 8/4-M2-PRINT
Page: 10 of 62
Change:
DESCRIPTION (Continued): the correct normalization and exponent after the rest
of the number has been assembled. The arithmetic in
this routine is double precision.

As each character is brought in, it is tested by a
series of subtractions to determine whether it is a
(1) digit, (2) + or - sign, (3) decimal point, (4) E
for exponent, or (5) a blank.

Exponents are sometimes indicated only by having plus
or minus signs occurring in the character sequence, e.g.,
+.1043 + 12 = .1043E + 12. If a sign is not noted
where one should occur it is assumed to be plus.
Switches are set to indicate whether the next sign is
for an exponent, whether a decimal point has been
encountered, and whether we are assembling the exponent,
cf., Fig. 7.

If the character read in is an integer, the previous
number assembled is multiplied (double precision) by
ten and the new integer is added to it. Exponents are
assembled in the same way only in a modifier. The
number of integers past a decimal point is combined
with the exponent to provide the final normalization
after the entire number has been assembled. Normaliza-
tion takes place by calling FEXP. The completed number
is in the accumulator in double precision.

FEXP

This is a double precision subroutine that makes the
final normalization in READEC and the initial normaliza-
tion in FDP. The exponent of 10 is in M4 when FEXP is
entered and the number being assembled is in the
accumulator. A table of 10, 10^2, 10^4, 10^8, 10^16 is
contained in the routine. If the ten's exponent is

Date: 7/16/64
Section: 8.4-M2-PRINT
Page: 11 of 62
Change:
DESCRIPTION (Continued): positive these powers of ten will be used to multiply the number in the accumulator; if the exponent is negative, they will be used for division.

The correct powers of ten are chosen in the following way: first \( 16 \) is successively subtracted from the exponent until the exponent is less than \( 16 \). Each time such a subtraction was possible the number is multiplied (divided) by \( 10^{16} \). When the exponent is less than \( 16 \) it will have a binary representation in \( M4 \) of \( \text{0000-1111} \) corresponding to \( 10^0 \) to \( 10^{15} \). Right shift of \( M4 \) puts the low-order bit into the sign bit of \( F5 \). If the sign bit is negative multiply (divide) by that power of ten; if positive ignore. When this is completed do another circular right shift, picking up the next higher power of ten in the table and repeat the above. The above process is carried out four times getting all powers of ten from 0 to 15. All multiplications and divisions are double precision. The normalized number is in the accumulator when finished.

FDP

This subroutine is used to punch or print double precision decimal numbers for the E, F or I formats. FDP uses the FEXP subroutine to normalize the numbers. The main problem this subroutine has is rounding since once a digit is output it can no longer be changed. Consecutive nines have to be saved until the proper rounding procedure is determined by the numbers following it. A particularly bad case, for example, is 9.9999 which may be rounded to 10.0000. However, before rounding can take place all the nines have to be converted. Also a leading blank space has to be saved to make room for the 1.
DESCRIPTION (Continued): The following is a description of the modifiers and the various sections of the program.

**Modifier Use**

MO GACVT puts converted digit -9 into M0. GA\$UTl moves digit to M4.

ML Set to -4096 if a blank is being held back for possible overflow of F field. Cleared if a zero is encountered during the GALD section, or when a digit is printed by GA\$UTl.

M2 Contains N at entry.

M3 Link.

M4 Set negative initially, indicating that there is no digit being saved. If GALD encounters a zero when ML is negative, M4 is set to this zero, thus preserving it for later printing.

It is used to preserve characters other than 9's for printing when the next non-nine character is converted. GA\$UTl prints this character (if it is nonnegative), and, before exiting, copies M0 + 9, the next non-nine character into M4.

M5 Minus the number of nonzero digits to be converted is set in here for counting.

M6 Is set to

0 ⇒ no sign

> 0 ⇒ plus sign

< 0 ⇒ minus sign

It is reset to 0 when the sign is printed.

If GALD section prints a point, it precedes it with a sign as necessary and clears M6. If M6 ≠ 0 when GA\$UTl is entered, a sign is printed.
DESCRIPTION (Continued):

M7 Is negative if the decimal point is to be inhibited. This is an entry parameter and is not damaged.

M8 Used as temporary storage initially to construct various counts. Then it is used to transmit characters to FLDBF.

M9 Contains k on entry. Minus the "lead-in" count is placed in here for use in GALD. This is the number of blanks plus the number of zeros which precede the number (-1 if an F field). If this count is negative, the field is too short and an alternative format of \( E + N \cdot (N - 6) \) is used if this is legal, where \( N \) is the input parameter in M2. For F fields, -1 is allowed and is changed to 0, but it means that there is no spare blank should the field be one too long because of rounding. If this occurs, the final 9 in the output is rounded up to *.

During GACVT and GAOUTL, M9 carries -1 minus the number of consecutive 9 digits encountered.

M10 Is set to Blank (0) initially. GALD prints this character. When a point is printed, it is changed to zero (10), thus giving zero suppression in front of the point. It is set to 9 for use by GAOUTL which prints \(-M9 - 1\) characters from M10 after printing M4. When the last digit has been converted, rounding may increase the number; in this case M10 is changed to zero (10).
DESCRIPTION (Continued):

ML1 Contains -1 minus the number of digits, zeros and blanks which precede the point. GALD and GAOUTL automatically insert the point when this count reaches 0.

ML2 Contains the scale factor s on entry. E fields do not change it, F fields and the decimal exponent of A to it, giving in both cases, the number of digits in front of the point.

ML3 Holds the exponent of the printed number.

ML4 Is not used.

ML5 Negative for E fields and ML5 bit 8 is a one if the sign digit is to be printed. This is an entry parameter and is not changed.

Sections of the Program FDP

First Section. Sets the sign code in M6. Scales the number in the accumulator by \(10^{-T}\) so that it is in the range \(1/10 \leq A < 1\). (It uses FEXP and GACVT for this.)

(Unless rounding increases the number to exactly 1, the placement of the number can now be made. To avoid finding \(\frac{1}{2} \times 10^{-n}\) for any n, rounding is deferred until the last digit is corrected. If this should change the output from 99 ... 9 to 10 ... 0, the following actions must be taken:

E increase the exponent by 1 and print 10 ... 0 instead (do not print extra digits).

F move the first digit one place to the left if possible. If this is not possible because there is no space, print 99 ... 9* instead.

The lead count is placed in M9, digit count in M5 and the point count in ML1. The lead
DESCRIPTION (Continued):

Count is the number of blanks and zeros to be printed in front of any nonzero digits. It is decreased by 1 in the case of F fields if possible, to allow for possible overflow by rounding. If M5, the number of nonzero digits to be printed, is negative, M9 and M11 must be reduced correspondingly. The point count is the number of blanks, zeros and digits to be printed before the point. If either the lead-in count or the point count is negative, an alternative standard format of E + N \cdot (N - 6) is used if N \geq 6; if N < 6, the exit to ERR\$RT is made.

Subroutines in FDP

GALD prints the lead zeros and blanks and inserts the point and sign if necessary.

GAVUTL is a subroutine that prints the digit saved in M4, if there is one, and the character from M10 -M9-1 times. It inserts a point if necessary, and prints a sign initially if it has not already been printed. If M1 is negative on entry, meaning that a space has been saved for an F field, this blank is printed first. M0+9 is sent to M4 and M9 is set to -1.

GACVT multiplies by 10 double precision and puts the integer part -9 in M0. If this is zero, M9 is decreased by 1 and exit is made to M3. Otherwise exit is made to M3+1.
<table>
<thead>
<tr>
<th>F4:</th>
<th>M0</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format Character Count</td>
<td>I/O List Word Address</td>
<td>Type: Input/Output Used for Field Length in E, F, I</td>
<td>Return</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5:</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Repeat or 1 Reentry Bit (0= + =&gt; Last 1= - =&gt; More)</td>
<td>Address in Core of Data Block</td>
<td>Number of Words</td>
<td>First Format Address</td>
<td></td>
</tr>
</tbody>
</table>

F5 filled from I/O List Word.

<table>
<thead>
<tr>
<th>F6:</th>
<th>M8</th>
<th>M9</th>
<th>M10</th>
<th>M11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format Character Symbol Read Temporary Count (Decimal Place Count for E, F Also)</td>
<td>Multiplicity of Current Format Instruction, e.g., 5 of 5E15,6</td>
<td>Repetition Count for Parenthesis</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7:</th>
<th>M12</th>
<th>M13</th>
<th>M14</th>
<th>M15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaling Power Address of Push Down Bits of Parenthesis For Interrogating Switches in M15 Switches in Low Bit Position. Also + =&gt; F, - =&gt; E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 => on, 0 => off.

Figure 1. Modifier Layout for Master Control Logic
M15 < 0, i.e., - ⇒ E type
M15 > 0, i.e., + ⇒ F or I type

starting from low bit positions. If a one is in this position, implication is true.

1 ⇒ (  
2 ⇒ P  
3 ⇒ )  
4 ⇒ -  
5 ⇒ + 
6 ⇒ .  
7 ⇒ /  
8 ⇒ *  
9 ⇒ Print if = 1, Punch if = 0 and switch 10 is off.  
10 ⇒ Input, i.e., card read.

Figure 2. Switches in M15 for Master Control Logic

<table>
<thead>
<tr>
<th>Representation--Actual Character</th>
<th>Representation--Actual Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 = blank</td>
<td>39 = P</td>
</tr>
<tr>
<td>1-9 = 1-9</td>
<td>40 = Q</td>
</tr>
<tr>
<td>10 = 0</td>
<td>44 = *</td>
</tr>
<tr>
<td>17 = /</td>
<td>48 = +</td>
</tr>
<tr>
<td>18 = s</td>
<td>49 = A</td>
</tr>
<tr>
<td>23 = X</td>
<td>52 = D</td>
</tr>
<tr>
<td>27 = ,</td>
<td>53 = E</td>
</tr>
<tr>
<td>28 = (</td>
<td>54 = F</td>
</tr>
<tr>
<td>32 = -</td>
<td>56 = H</td>
</tr>
<tr>
<td>35 = L</td>
<td>57 = I</td>
</tr>
<tr>
<td>36 = M</td>
<td>60 = )</td>
</tr>
</tbody>
</table>

Figure 3. BCD Characters
### F4: 

<table>
<thead>
<tr>
<th>MO</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left, Right Switch for Correct 1/2 of 1/4 Word</td>
<td>Format Character Count</td>
<td>1/4-Word Count in Format Word</td>
<td>Current Format Word Address</td>
</tr>
</tbody>
</table>

Figure 4. FRCNT

### F5: 

<table>
<thead>
<tr>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Buffer Character Count -133 or -80</td>
<td>Address in Buffer Where FHFWD Is to Be Placed</td>
<td>Left or Right 1/2 or 1/4 Word (-1 \Rightarrow \text{Left}, \ 0 \Rightarrow \text{Right})</td>
<td>Quarter Word Count</td>
</tr>
</tbody>
</table>

Figure 5. FCNTS

### F5: 

<table>
<thead>
<tr>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Character Count -80</td>
<td>Address of Current Word in Input Buffer</td>
<td>Right or Left 1/2 of Quarter Word. (-1 \Rightarrow \text{Left}, \ 0 \Rightarrow \text{Right})</td>
<td>Quarter Word Count</td>
</tr>
</tbody>
</table>

Figure 6. FCNTSC
<table>
<thead>
<tr>
<th>F4:</th>
<th>M0</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not Used</td>
<td>Number of Places Past Decimal Point</td>
<td>Length of Field</td>
<td>Return Address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F5:</th>
<th>M4</th>
<th>M5</th>
<th>M6</th>
<th>M7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exponent in Here</td>
<td>Last Character Read</td>
<td>Original Return Saved When Using FRDBF or FEXP</td>
<td>Length of Field</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F6:</th>
<th>M8</th>
<th>M9</th>
<th>M10</th>
<th>M11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Character Read</td>
<td>= 1 ⇒ Next Sign for Exponent = 0 Initially</td>
<td>0 ⇒ Exponent Plus -1 ⇒ Exponent Minus</td>
<td>+1 ⇒ Number + -1 ⇒ Number -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F7:</th>
<th>M12</th>
<th>M13</th>
<th>M14</th>
<th>M15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>= 1 ⇒ in Exponent = 0 Initially</td>
<td>= 1 ⇒ Digit Read is First Digit of Number</td>
<td>= 1 ⇒ Digit Read is the First Digit of the Exponent</td>
<td>= 1 ⇒ Decimal Point Has Been Encountered</td>
</tr>
</tbody>
</table>

Figure 7. READEC Modifiers
MAIN FORMAT SCAN CONTROL
PART 1
Main Format Scan Control

Part 2
FBACK

SAVE F4 AND F7

LOGIN ADDRESS OF LAST UPTaked MACRO.
THESS. PUT MACRO WORD, SPECIFIED INTO F7

PUT RUNNING COUNTS AND SWITCHES, FRGMT, INTO F4

PICK UP PROPER FORMAT CHARACTER

RESET FRGMT

SAVE FRGMT

RESTORE F4, FT AND RETURN

FBACK
READEC

SAVE FAST REGISTERS, INITIALIZE SWITCHES AND COUNTERS.

READ NEXT CHARACTER (CALL FROBF).

END OF FIELD?

YES

BLANK?

NO

NUMBER?

NO

+ OR -?

NO

NO

YES

COMBINE NUMBER OF DECIMAL PLACES WITH EXPONENT

CALL FEXP TO NORMALIZE THE NUMBER, SETS EXPONENT

RESTORE REGISTERS AND RETURN

ADD TO 10^EXPONENT IN M4.

INCREMENT DECIMAL COUNT

YES

ERROR

NO

MAKE SIGN+, NEXT FOR EXPONENT

SET SIGN FOR EXPONENT

SET SIGN FOR EXPONENT

SET SIGN FOR NUMBER, NEXT FOR EXPONENT

SET EXPONENT INDICATOR IF NOT SET ALREADY

SET DECIMAL POINT INDICATOR

ADD NEW DIGIT TO 10# PREVIOUS DIGITS READ.

NO

AFTER DECIMAL PT.

YES

E = EXPONENT

NO

YES

IN EXPONENT?

NO

IN EXPONENT?

YES

SIGN READ ALREADY?

NO

= DECIMAL POINT?

NO

YE
ENTRY PRINT,READ,PUNCH
READEC CAD 15,3, SUBROUTINE TO READ IN DECIMAL NUMBERS
SFR 4, FCOM29 SET ACCUMULATOR TO 0
SFR 5, FCOM26 SAVE FAST REGISTERS
SFR 6, FCOM27
SFR 7, FCOM28
CAM 1, M9 PLACES PAST DEC. PT.
CAM 4
LFR 6, ZERO M12=M15=0, M13=M14=1
LFR 7, CON
JZM 2, ERROR
CSM 7, M2 FIELD LENGTH
XX CALL FRDBF
SUBRT JZM 8, X BLANK
ADM 8,-11
JNM 8, NUMBER
YYY ADM 8,-37
JZM 8, PLUS
ADM 8, 16
JZM 8, MINUS
SBM 8, 21
JZM 8, EXP
SBM 8, 6
JUM 8, ERROR DEC. PT.
DECPT JUM 15, ERROR DEC. PT. INDICATOR SET ERROR
CAM 15, 1 SET DEC. PT. INDICATOR
CAM 1
TRA X
PLUS JUM 12, A IS THIS SIGN FOR EXPONENT
JUM 9, A
CAM 11, 1 SET PLUS FOR NUMBER
TRA C
MINUS JUM 12, B IS THIS SIGN FOR EXPONENT
JUM 9, B
CSM 11, 1 SET MINUS FOR NUMBER
C    CAM   9,1    SET NEXT SIGN FOR EXPONENT
      CAM   13
X    CJU   7,XX    ENTIRE NUMBER IN
COMPNO LFR   7,FCOM28    NORMALIZE NUMBER
       JPM   15,J5    COMBINE EXPONENT AND NUMBER OF
       CAM   12    PLACES AFTER .
J5    JPM   10,J6
       CSM   4,M4    NORMALIZE WITH PROPER EXPONENT
J6    SBM   4,M12+M1    CALL FEXP
       CALL FEXP
END   LFR   4,FCOM29    PLUS
       JPM   11,U    MAKE NUMBER MINUS
       STC   2,3
       STN   0,3
       SUB   2,3
U    LFR   6,FCOM27    RESTORE FAST REGISTERS
       LFR   5,FCOM26
       LFR   7,FCOM28
       JLH   3,0    RETURN
B    CSM   10,1    SET EXPONENT MINUS
A    JZM   14,ERROR    IF EXPONENT SIGN ALREADY IN, ERROR
D    CAM   14    SET EXPONENT SIGN IN
D1   CAM   12,1    SET EXPONENT INDICATOR
      TRA   X
EXP   JUM   12,ERROR    IF ALREADY IN EXPONENT, ERROR
      TRA   D1
      NUMBER CJU   8,NUMBER+1    ZERO = 10 BCD, MUST BE ADJUSTED
      CAM   8,-10
      JUM   12,EXPR    IS THIS NUMBER IN EXPONENT
      JZM   13,F    SIGN OF NUMBER IN
      CAM   11,1    IF NO SIGN, MAKE PLUS
      CAM   9,1    NEXT SIGN FOR EXPONENT
      CAM   13    SET SIGN IN
F    STC   2,3    DOUBLE PRECISION ARITHMETIC
      MPY   10    10 X PREVIOUS DIGITS
XCH   2,3,
MPY   100.
ADD   2,3
ADD   10+M8.
JZM   15,X
ADM   1,1
TRA   X
EXPRT JZM   14,H
CAM   10,1
CAM   14
H
CRM   4,10
CRM   4,3
ADM   4,10+M8
JDC   0,X
FIL
ZERO OCTQ 00000,00000,00000,00000
CON OCTQ 00000,00001,00001,00000
ASSIGN FCOM26,FCOM27,FCOM28,FCOM29
FDP SFR 4,GAT
SFR 5,GAT+1
SFR 6,GAT+2
SFR 7,GAT+3
CAM  6
TZ    GA22A
STC   F2
XCH   F2
TIP   GA1
STN   F0
SSC   F2
CAM   6,4096
XCH   F2
GA1   STR F3
SEX   3
CAD   M3
MPY   GAT+4

PLUS NEW DIGIT
AFTER DEC. PT.
INCREMENT DECIMAL COUNT
IF NO SIGN FOR EXPONENT, SET PLUS
SET EXPONENT SIGN IN
10 X PREVIOUS EXPONENT DIGITS
ADD NEW DIGIT
SAVE FAST REGISTERS
STORE NUMBER
POSITIVE
CHANGE SIGN
SET NEGATIVE INDICATOR
EXponent = LOG BASE 10 OF 4
SIA  8  =T-1
CAM 4,M8-1 -T
CAD  F2
ADD  F3
CALL FEXP  A=10**(T)
STC  F3
STR  F2  STORE AWAY
ADD  F3
SUB  GAT+5
SUB  GAT+6  A-1/10
TZP  GA2  POSITIVE OR ZERO
CAD  F3
CALL GACVT  A=10
GAT1  
BSS  1
ADD  M0+9
ADM  8,-1  T1=T-1
STR  F3

g2  CAM  1
ANN  15,16
ORM  6
JPM  15,G8
CAM  13,M8+1-M12
G26  CAM  8,3
G9  ADM  8,M9-M2+M12  LEAD COUNT
JPM  7,G3
ADM  8,1
G3  JZM  6,G4  NO SIGN
ADM  8,1
G4  CAM  11,M8-M12  POINT COUNT
CAM  5,M12+M9  DIGIT COUNT
JPM  5,G5
SBM  8,M5  POSITIVE
G5  CSM  5,M5  DECREMENT LEAD COUNT
CAM  9,M8  -1-LEAD COUNT TO M9
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMP 9,GAL25</td>
<td>LEAD COUNT NEGATIVE</td>
</tr>
<tr>
<td>JMP 11,GAL25</td>
<td>POINT TOO FAR TO LEFT</td>
</tr>
<tr>
<td>JNM 15,GAL6</td>
<td>E FIELD</td>
</tr>
<tr>
<td>CJZ 9,GAL7</td>
<td>SAVE BLANK FOR F FIELD</td>
</tr>
<tr>
<td>CAM 1,4096</td>
<td>SET SAVED</td>
</tr>
<tr>
<td>GA6 CAM 10</td>
<td>LEAD CHARACTER BLANK</td>
</tr>
<tr>
<td>CAM 4,4096</td>
<td>NO DIGIT YET</td>
</tr>
<tr>
<td>Gald CJU 11,GALD3</td>
<td>NOT TIME FOR POINT</td>
</tr>
<tr>
<td>Jzm 6,GALD2</td>
<td>NO SIGN</td>
</tr>
<tr>
<td>CAM 8,32</td>
<td>MINUS</td>
</tr>
<tr>
<td>JNM 6,GALD1</td>
<td>CORRECT</td>
</tr>
<tr>
<td>CAM 8,48</td>
<td>PLUS</td>
</tr>
<tr>
<td>Gald1 CALL</td>
<td>OUTPUT SIGN</td>
</tr>
<tr>
<td>CAM 6</td>
<td>CLEAR SIGN INDICATOR</td>
</tr>
<tr>
<td>Gald2 JPM 7,GALD5</td>
<td>NO POINT</td>
</tr>
<tr>
<td>CAM 8,59</td>
<td>OUTPUT POINT</td>
</tr>
<tr>
<td>Call FLDBF</td>
<td>DONE ON LEAD CHAR</td>
</tr>
<tr>
<td>Gald5 CJZ 9,GALD6</td>
<td>CHARACTER NOW ZERO</td>
</tr>
<tr>
<td>Cam 10,10</td>
<td>E FIELD</td>
</tr>
<tr>
<td>JNM 15,GALD4</td>
<td>FIRST DIGIT ZERO</td>
</tr>
<tr>
<td>Cam 4</td>
<td>NO SPARE</td>
</tr>
<tr>
<td>JPM 1,GALD3</td>
<td>CLEAR SPARE INDICATOR</td>
</tr>
<tr>
<td>Cam 1</td>
<td>DONE ON LEAD CHARACTERS</td>
</tr>
<tr>
<td>Gald3 CJZ 9,GALD6</td>
<td>OUTPUT LEAD CHARACTER</td>
</tr>
<tr>
<td>Gald4 Cam 8,M10</td>
<td>LOOP</td>
</tr>
<tr>
<td>Call FLDBF</td>
<td>NO SPARE IN F FIELD</td>
</tr>
<tr>
<td>G7 Tra GALD</td>
<td>Q=S+T</td>
</tr>
<tr>
<td>Cam 9,-1</td>
<td></td>
</tr>
<tr>
<td>Tra GALD</td>
<td></td>
</tr>
<tr>
<td>G8 Adm 12,M8+1</td>
<td></td>
</tr>
<tr>
<td>Cam 8,-1</td>
<td></td>
</tr>
<tr>
<td>Tra GA9</td>
<td></td>
</tr>
<tr>
<td>Gald6 Cam 9,-1</td>
<td></td>
</tr>
<tr>
<td>Cam 10,9</td>
<td></td>
</tr>
</tbody>
</table>
JPM 5, GA27
CALL GACVT
CJU 5, GA10
TRA GA18
CALL GAOUT1
CJU 5, GA10
SUB 10, 3, 2048
TN GA19
ADM 4, 1

CALL GAOUT1
JPM 15, GA17
F TYPE

CALL FLDBF
CAM 8, 53
JPM 13, GA12
CAM 8, 48
EXponent SIGN

GA12 CALL FLDBF
CAM 8
ADM 13, -10
JNM 13, GA14

GA13 CJU 8, GA13
CAM 8, 10

CALL FLDBF
CAM 8, M13+10
JUM 8, GA16
CAM 8, 10

GA15 CALL FLDBF
CAM 8, 10

LFR 4, GAT
LFR 5, GAT+1
LFR 6, GAT+2
LFR 7, GAT+3

GA16 CALL FLDBF
JUM 8, GA17

LFR 4, GAT
LFR 5, GAT+1
LFR 6, GAT+2
LFR 7, GAT+3

J LH M3

GA18 SUB 10, 3, 2048
DO WE ROUND LAST 9
TN   GA19
CAM  10,10
JPM  4,GA20
JPM  15,GA21
ADM  9,1
ADM  13,1
GA22 CAM 4
GA20 ADM 4,1
TRA  GA19
JPM  1,GA24
CAM  1
TRA  GA22
GACVT MPY 10,
STC  F3
XCH  F2
MPY  10,
ADD  F3
ASC  F2
XCH  F2
TZP  GCVT9A
CAD  15,3,
STR  F2
GCVT9A SIA 0
SUB  M0,
ADM  0,-9
JNM  0,M3+1
JZM  0,GCVT8A
ADD  1,
CAM  0
GCVT8A ADM 9,-1
JLH  M3
GAOUT1 SFR 4,GAT1
JNM  1,GOUT6
JZM  6,GOUT2
CAM  8,32

NO
ZERO IN M10
PREVIOUS NON 9 DIGIT
F FIELD
INCREASE EXPONENT
SET ZERO AS PREVIOUS DIGIT
ROUND UP
OUTPUT 10..0
NO SPACE SAVED
USE SPACE FOR 1

DOUBLE PRECISION MULTIPLY BY 10
MOST SIGNIFICANT PART IN ACC.
CALL GAOUT1
TRA GA19

GA25 CAM 9,M2-6
JNM 9,ERROR
CAM 7,4096
ORM 6,32
CAM 13,M12
CAM 12
ORM 15,4096
TRA GA26

GA27 JPM 4,GA30
JNM 15,GA28
JPM 1,GA17
CAM 1
CAM 4

GA30 SUB 10,3,2048
TN GA19
JUM 5,GA19
ADM 4,1
TRA GA19

GAT BSS 4
OCTQ 04642,00465,0522,14000
OCTQ 03146,06314,14631,11577,03146,06314,14631,11551

FEXP SFR 5,FEXP1
SFR SUBROUTINE TO NORMALIZE DECIMALS NUMBERS
LFR 3,FTABL
JNM 4,FEXP9
JUM 2,3
TRA FEXP1

FEXP2 MPY 3,3
XCH 2,3
MPY 3,3
ASC 2,3

FEXP1 SBM 4,16
CSM 5,4
JPM 4,FEXP2

TEMPORARY STORAGE

DIGIT ALLREADY CONVERTED
E TYPE, PRINT EXPONENT
NO SPARE
CLEAR SPARE

DIGIT IS ZERO
NO ROUNING
NO ROUNING TOO SMALL
ROUND

EXPONENT GREATER THAN 16
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEXP3</td>
<td>CRM 4,1</td>
</tr>
<tr>
<td></td>
<td>JPM 4,FEXP4</td>
</tr>
<tr>
<td></td>
<td>MPY 5,FTABL</td>
</tr>
<tr>
<td></td>
<td>STC 0,3</td>
</tr>
<tr>
<td></td>
<td>CAD 1,3</td>
</tr>
<tr>
<td></td>
<td>MPY 2,3</td>
</tr>
<tr>
<td></td>
<td>ADD 0,3</td>
</tr>
<tr>
<td></td>
<td>STC 2,3</td>
</tr>
<tr>
<td>FEXP4</td>
<td>CJU 5,FEXP3</td>
</tr>
<tr>
<td></td>
<td>ADD 2,3</td>
</tr>
<tr>
<td></td>
<td>TRA FEXP10</td>
</tr>
<tr>
<td>FEXP9</td>
<td>SBM 4,1</td>
</tr>
<tr>
<td></td>
<td>TRA FEXP5</td>
</tr>
<tr>
<td>FEXP6</td>
<td>DIV 3,3</td>
</tr>
<tr>
<td></td>
<td>SRM 2,3</td>
</tr>
<tr>
<td></td>
<td>XCH 2,3</td>
</tr>
<tr>
<td></td>
<td>DIV 3,3</td>
</tr>
<tr>
<td></td>
<td>ADD 2,3</td>
</tr>
<tr>
<td>FEXP5</td>
<td>ADM 4,16</td>
</tr>
<tr>
<td></td>
<td>JNM 4,FEXP6</td>
</tr>
<tr>
<td></td>
<td>CSM 5,4</td>
</tr>
<tr>
<td>FEXP7</td>
<td>CRM 4,1</td>
</tr>
<tr>
<td></td>
<td>JNM 4,FEXP8</td>
</tr>
<tr>
<td></td>
<td>DIV 5,FTABL</td>
</tr>
<tr>
<td></td>
<td>SRM 2,3</td>
</tr>
<tr>
<td></td>
<td>STR 0,3</td>
</tr>
<tr>
<td></td>
<td>CAD 1,3</td>
</tr>
<tr>
<td></td>
<td>VID 2,3</td>
</tr>
<tr>
<td></td>
<td>ADD 0,3</td>
</tr>
<tr>
<td>FEXP8</td>
<td>CJU 5,FEXP7</td>
</tr>
<tr>
<td>FEXP10</td>
<td>LFR 5,FEXPC1</td>
</tr>
<tr>
<td></td>
<td>JLH M3</td>
</tr>
<tr>
<td>FEXPC1</td>
<td>BSS 1</td>
</tr>
<tr>
<td>OCTQ</td>
<td>05000,00000,00000,00002</td>
</tr>
<tr>
<td>OCTQ</td>
<td>03100,00000,00000,00004</td>
</tr>
</tbody>
</table>
OCTQ 04704.00000.00000.00007 10 FOURTH
OCTQ 02765.16040.00000.00016 10 EIGHTH
FTABL OCTQ 04361.13623.07701.00033 10 SIXTEENTH
READ SFR 7,FCOM4
LFR 7,FCOM1
JNM 12,RD1
SFR 7,FCOM1
CAM 15,512
CALL FRESET
TRA 81
REPEAT ENTRY
RD1 LFR 7,FCOM3
SFR 6,FCOM3
LFR 6,FCOM2
SFR 5,FCOM2
ATN 1,1
LFR 5
LDM 7,FCOM1
SFR 4,FCOM1
LFR 4,FT6
JLH M3
NEW I-O LIST WORD
PUNCH SFR 7,FCOM4
LFR 7,FCOM1
JNM 12,RD1
CAM 12,.80
CAM 13
CAM 14,.2
CAM 15,M
TRA 82
REPEAT ENTRY
PRINT SFR 7,FCOM4
LFR 7,FCOM1
JNM 12,RD1
CAM 12,.133
CAM 13,256
CAM 14,.1
CAM 15,2+M
COUNT FOR PULLED CHARACTERS
REPEAT ENTRY
COUNT FOR PRINTED CHARACTERS
B2  SFR  4,FCOM1
     CAM  3,M13
     CAM  13,OUTBF
     SFR  7,FCNTS
     CAM  15,M3
B1  SFR  6,FCOM3
     LDM  11,M1
     CAM  8,2
     CAM  9
     CAM  10,-4
     SFR  6,FRCNT
     CAM  12
     SFR  6,PUSHDN+2
     CAM  13,PUSHDN
     SFR  5,FCOM2
     ATN  1,1
     LFR  5
     BLPR  CAM  10,M11
          CAM  11,M9-1
          LDM  9,FRCNT
          CAM  8,M12
          SBM  13,PUSHDN+4
          JPM  13,FRMERR
          ADM  13,PUSHDN+5
     SFR  6,M13
     AF2  CAM  7
     AF3  CAM  9
     AF5  CALL  FORMRD
     AF1  JZM  8,AF5
          ADM  8,-44
          JZM  8,BSTAR
          ADM  8,-16
          JZM  8,BPR
          ADM  8,43
          JZM  8,BSLASH
     OUTPUT CONTROL WORD
     SET PRINT OR PUNCH CONTROL
     FORMAT START ADDRESS
     FORMAT CONTROL WORD SET
     SET SCALE TO ZERO
     INITIALIZE PARENTHESIS PUSH DOWN
     M13 IS PUSH DOWN ADDRESS
     FIRST CONTROL WORD
     LEFT PARENTHESIS AND START OF FORMAT
     MAKE FIRST PUSH DOWN ENTRY
     TOO MANY NESTED PARENTHESIS
     STORE ENTRY
     M7 IS USED TO DETERMINE WHICH CHAR. MAY BE
     READ. NUMBERS ARE CONSTRUCTED IN M9
     READ FORMAT CHAR.
     SKIP BLANK CHARACTER
     ASTERISC
     RIGHT PARENTHESIS
     SLASH. END OF LINE
ADM 8,10
JZM 8,AF2
JNM 7,FRMERR
ADM 8,16
JSB 0,BSUB2
FIL
ANN 7,2
CAM 14
JUM 14,AF4
ADM 8,7
JZM 8,BS
ADM 8,5
JZM 8,BX
ADM 8,34
ANM 15,4077
JZM 8,BEF
ADM 8,3
JZM 8,BEF
ORM 15,4096
CJZ 8,BEF
ADM 8,3
JZM 8,BH
ADM 8,7
CAM 2
JZM 8,BA
ADM 8,3
JZM 8,BD
CJZ 8,BA
ADM 8,11
JZM 8,BQ
ADM 8,4
JZM 8,BM
CJZ 8,BL
ADM 8,7
JZM 8,BLPR

COMMA
SHOULD HAVE BEEN TERMINATING CHAR.
DIGIT, GO CONVERT
MUST BE P SINCE + OR - OCCURRED
S FOR SPACES
X FOR SPACE
SET INHIBIT PLUS AND F SWITCHES
F FIELD
I WHICH IS SAME
SET E SWITCH
E FIELD
HOLLERITH FIELD
A FIELD
D FIELD
Q FIELD FOR OCTAL QUARTER WORDS
M FIELD FOR OCTAL LOCATIONS
L FIELD FOR DECIMAL LOCATIONS
LEFT PARENTHESIS
BEF  CALL FORMRD
CAM  14,M8-37
CAM  0,8E2
JZM  14,BSUB4
ADM  14,26
JNM  14,BSUB3
ADM  14,-37
EOM  15,2
JUM  14,BEF1
EOM  15,18
BEF1 JSB  0,BSUB
  FIL
BEF2 JPM  7,8E3
BEF7A CAM  0,M8+11
  FIL
BEF7 CALL BINC
JZM  6,8E4
ANN  15,512
CAM  14
JUM  14,8E5
CAD  5,1
CRN  15,2
CAM  14
JPM  14,BEA1
ADD  5,1
BEA1 CALL FDP
BEF6 ADM  6,-1
ADM  10,-1
JPM  10,BE7
CAM  12
SET TERMINATING CHARACTER REQUIRED
NEXT CHARACTER TO M8
RETURN TO FORMAT TEST
INPUT NEXT NUMBER
STORE MOST SIGNIFICANT HALF
DOUBLE PRECISION BIT
BRANCH IF SINGLE PRECISION
STORE LEAST SIGNIFICANT HALF
GO TO END TESTS
GET NEXT IO CONTROL WORD
ASTERISK, CHECK FOR EMPTY PUSHDOWN
STILL INSIDE PARENTHESIS
SET PUSHDOWN TO LAST LEFT PARENTHESIS (OUTSIDE)
FORMAT ADDRESS OF LAST LEFT PARENTHESIS
ZERO IF NO PARENTHESSES ENCOUNTERED
SET PUSHDOWN TO BEGINNING OF FORMAT
RESET PERENTHESES REPEAT COUNT
NEW LINE ON RETURN AND GO TO BACK UP
LAST CONTROL WORD
LOAD NEXT CONTROL WORD
REPEAT ENTRY BIT ON
JNM 14, BE9A
CALL FRESET
BE9A
SFR 4, FT6
LFR 4, FCOM1
SFR 5, FCOM1
LFR 5, FCOM2
SFR 6, FCOM2
LFR 6, FCOM3
SFR 7, FCOM3
LFR 7, FCOM4
JLH 3,
BE3
CAM 14, M8-48
CAM 2, M9
CAM 9
JUM 14, BE7A
CAM 7, 4096
TRA BE1
BS
CAM 10, M9-1
CAM 9
JSB 0, BSUB
FIL
CAM 0, M8+11
JZM 9, BS7
BX1
CRN 15, 10
CAM 7
CAM 8
BS6
SFN 9, v
CAM 2
JNM 7, BS4
BS5
CALL FLDDBF
CJU 2, BS5
BS3
ADM 10, -1
JPM 10, BS6
TRA BS7
BS4
CALL FRDBF

INPUT TYPE
OTHERWISE PRINT LATS LINE

RESTORE F4
SAVE F5 FOR REPEAT ENTRY
RESTORE F5
SAVE F6
RESTORE F6
SAVE F7
RESTROE F7
EXIT

LENGTH COUNT TO M2
CLEAR POINT COUNT
JUMP IF NOT POINT
SET POINT READ

SPACES S
CLEAR NUMBER
READ NUMBER

SAVE NEXT DIGIT
NO SPACES

INPUT OUTPUT BIT
BLANK TO M8

MINUS COUNT F6R S
INPUT
OUTPUT ~MI BLANKS

COUNT REPEAT COUNT

READ ~M2 CHARACTERS
BH

CJU 2, BS4
TRA BS3
BH
SFML 9,
CAM 2
JZM 9, BH6
CRN 15, 10
CAM 14
JNM 14, BH2
BH1 CALL FORMRD
CALL FLDBF
CJU 2, BH1
TRA BH6

BH2 SFR 5, FCOM12
LFR 5, FRCNT
SFR 7, FCOM13
BH5 CALL FRDBF
CAM 9, 63
CJZ 4, BH3
CRM 8, 7
CRM 9, 7

BH3 LFR 7, M7
JLH BH3A+M6+M6+8
FIL

BH3A NAM 12, M9
ORM 12, M8
JZM 4, BH4
TRA BH3B
NAM 13, M9
ORM 13, M8
JZM 4, BH4
TRA BH3B
NAM 14, M9
ORM 14, M8
JZM 4, BH4
TRA BH3B

HOLLERITH FIELD, INVERT COUNT
ZERO COUNT
INPUT OUTPUT BIT
INPUT
READ FORMAT CHARACTER
OUTPUT
COUNT
SAVE F5
FORMAT CONTROL WORD
SAVE F7
READ CHARACTER
SET MASK
LEFT OR RIGHT, BRANCH IF RIGHT
MOVE TO LEFT
DITTO MASK
CURRENT FORMAT WORD
BRANCH ACCORDING TO QUARTER WORD
FIRST QUARTER
CHARACTER REPLACED IN FORMAT
RIGHT HAND PART
SECOND QUARTER DITTO
THIRD QUARTER DITTO
NAM 15,M9
ORM 15,M8
JUM 4,BH3B
ADM 7,1
CAM 4,-2
CAM 6,-4
ATN -1
BH3B SFR 7,M7
ADM 5,1
CJU 2,BH5
SFR 5,FRCNT
LFR 5,FCOM12
LFR 7,FCOM13
BH6 JPM 7,AF2
BA3 ADM 5,1
ADM 6,-1
JPM 12,BA5
ADM 10,-1
CAM 9,M0
JPM 10,BA4
LFR 7,FCOM6
SFR 7,FRCNT
LFR 7,FCOM5
CAM 0,M14
TRA BS7
BA5 CAM 9,M12+1
TRA BA4
BH4 CAM 4,-2
ADM 6,1
TRA BH3B
BSUB1 CRM 9,10
CRN 9,2
ADM 9,M8+11
CJU 8,BSUB
ADM 9,-10

FOURTH QUARTER

JUMP IF LEFT HAND PART
INCREMENT WORD ADDRESS
RESET FORMAT CONTROL
STORE CURRENT FORMAT WORD BACK IN MEMORY
INCREASE FORMAT ADDRESS
BY ONE. THIS IS A COUNT OF THE NUMBER OF FORMAT
CHARACTERS. COUNT NUMBER OF HOLLERITH CHARACTER
S. RESTORE FORMAT COUNT, F5 AND F7

THIS IS POSITIVE IF HOLLERITH, -VE IF A FIELD
INCREMENT 10 LIST ADDRESS
DECREASE 10 COUNT
TEST FOR MORE A CHARACTERS
TEST REPEAT COUNT
RESTORE M9 TO NUMBER FOLLOWING A
POSITIVE IF MUST REPEAT
REPLACE TRUE FORMAT ADDRESS
CONTROL WORD
RESTORE F7
NEXT CHARACTER TO M0
END OF ALL A FIELDS FOR NOW
M9 CONTAINS NUMBER OF NEXT GROUP OF 8 OR LEES CH
ARACTERS
RESET TO LEFT HAND QUARTER
INCREASE QUARTER WORD COUNT

MULTIPLY BY TEN
AND ADD NEXT DIGIT
TEST FOR ZERO
RECODE FROM 12 BASE8
BSUB  CALL  FORMRD  READ NEXT FORMAT CHARACTER
BSUB3 JZM  8,BSUB  SKIP BLANK CHARACTER
ADM   8,-11
BSUB2 JNM  8,BSUB1 DIGIT
ADM   8,-26
JZM  8,BSUB4 N+, GOT READ A NUMBER FROM THE IO LIST
ADM   8,26
JLM   0,9 OTHERWİSE EXIT
BSUB4 SFR  4,FCOM6
CALL  BINC
LFR   4,FCOM6
FIL
RENN JUM   6,RENL SUBROUTINE WHICH PUTS M8 IN OUTPUT BUFFER AS NEXT CHARACTER
CAM   3,RENN
JPM   4,DE9
REN1 CAD   5,1, SUBROUTINE WHICH PUTS M8 IN OUTPUT BUFFER AS NEXT CHARACTER
ADM   6,-1
SIA    8
CAM   9,M8
TRA   BSUB
FLDBF SFR  5,FCOM7 BUFFER PREPARATION WORD
SFR    6,FCOM8 CONTROL WORD
SFR    7,FCOM9 TOO MANY CHARACTERS FOR LINE OR CARD
LFR    7,FBEWD BRANCH ACCORDING TO QUARTER
LFR    5,FCNTS FIRST QUARTER, CHAR. TO M12
JPM    4,FERRPR BRANCH IF RIGHT HAND PART
JLM    7,9 MOVE TO LEFT
FIL
M ADM   12,M8 SECOND QUARTER
CJZ    6,FN3A
CRM    12,7
TRA   FN8
ADM   13,M8
CJZ    6,FN3A
CRM   13,7
THIRD QUARTER

FOURTH QUARTER

STORE BUFFER PREPARATION WORD IN BUFFER

PREPARATION WORD TO BLANKS
RESET M7, QUARTER WORD CONTROL
RESET LEFT/RIGHT CONTROL
INCREMENT QUARTER CONTROL
INCREASE CHARACTER COUNT
STORE PREPARATION WORD

EXIT
SUBROUTINE TO BACK UP FORMAT CONTROL WORD TO
LAST LEFT PARENTHESIS IN CASE OF REPEAT
PUSHDOWN ENTRY
CONTROL WORD

RESET FORMAT WORD ADDRESS

LEFT RIGHT SWITCH RESET
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM 2</td>
<td>QUARTER WORD COUNT RESET</td>
</tr>
<tr>
<td>CRM 2,1</td>
<td>STORE BACK IN MEMROY</td>
</tr>
<tr>
<td>SBM 2,4</td>
<td></td>
</tr>
<tr>
<td>SFR 4,FRCNT</td>
<td></td>
</tr>
<tr>
<td>LFR 7,FT5</td>
<td></td>
</tr>
<tr>
<td>LFR 4,FCOM12</td>
<td></td>
</tr>
<tr>
<td>JLH 3,9</td>
<td></td>
</tr>
<tr>
<td>FORMRD SFR 4,FCOM12</td>
<td></td>
</tr>
<tr>
<td>SFR 7,FCOM13</td>
<td></td>
</tr>
<tr>
<td>LFR 4,FRCNT</td>
<td></td>
</tr>
<tr>
<td>LFR 7,M3</td>
<td></td>
</tr>
<tr>
<td>ORB 2,4</td>
<td></td>
</tr>
<tr>
<td>CAM 8,M12</td>
<td></td>
</tr>
<tr>
<td>CJZ 0,FORM2</td>
<td></td>
</tr>
<tr>
<td>CRM 8,6</td>
<td></td>
</tr>
<tr>
<td>FORM3 ADM 1,1</td>
<td></td>
</tr>
<tr>
<td>ANM 8,63</td>
<td></td>
</tr>
<tr>
<td>SFR 4,FRCNT</td>
<td></td>
</tr>
<tr>
<td>LFR 4,FCOM12</td>
<td></td>
</tr>
<tr>
<td>LFR 7,FCOM13</td>
<td></td>
</tr>
<tr>
<td>JLH 3,9</td>
<td></td>
</tr>
<tr>
<td>FORM2 CAM 0,2</td>
<td></td>
</tr>
<tr>
<td>CJU 2,FORM3</td>
<td></td>
</tr>
<tr>
<td>CAM 2,4</td>
<td></td>
</tr>
<tr>
<td>CAM 2,4</td>
<td></td>
</tr>
<tr>
<td>ADM 3,1</td>
<td></td>
</tr>
<tr>
<td>FORM3 TRA</td>
<td></td>
</tr>
<tr>
<td>FRESET ANN 15,512</td>
<td></td>
</tr>
<tr>
<td>CAM 14</td>
<td></td>
</tr>
<tr>
<td>SFR 5,FCOM7</td>
<td></td>
</tr>
<tr>
<td>SFR 4,FCOM24</td>
<td></td>
</tr>
<tr>
<td>SFR 7,FCOM9</td>
<td></td>
</tr>
<tr>
<td>JUM 14,FN00UT</td>
<td></td>
</tr>
<tr>
<td>LFR 5,FCNTS</td>
<td></td>
</tr>
<tr>
<td>FRST1 LFR 7,F8FWD</td>
<td></td>
</tr>
<tr>
<td>TRA FBLNK2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BRANCH IF INPUT</td>
</tr>
</tbody>
</table>
FBLNK1 ATN 5,1
SFR 7
LFR 7,FZERO
ADM 4,8

FBLNK2 JMN 4,FBLNK1
LFR 7,FCOM9
ANN 15,256
CAM 14
JUM 14,FN9A
CALL SYSIO
DECO WRITE+1,OUTBF,0,0
CAM 0,-80
CAM 2,-2
CAM 3,M
TRA FN9B

FN9A CALL SYSIO
DECO WRITE+2,OUTBF,0,0
CAM 0,-133
CAM 2,-1
CAM 3,2+M

FN9B CAM 1,OUTBF
SFR 4,FCNTR
LFR 7,FZERO
SFR 7,FBFW
TRA FRSTNC

FNOUT CALL SYSIO
DECO 1,INPBF,0,0
CAM 0,-80
CAM 2,-2
CAM 3,-4
CAM 1,INPBF
SFR 4,FCNTRSC

FRSTNC LFR 5,FCOM7
LFR 4,FCOM24
LFR 7,FCOM9

FILL OUT BUFFER WITH PREPARATION WORD, THEN
A
ALL BLANKS
NONE WORDS TO GO
PUNCH/PRINT BIT
OUTPUT TO PUNCH VIA SYSTEM
RESET BUFFER CONTROL WORD FOR PUNCH
PRINT VIA SYSTEM PROGRAM
RESET BUFFER CONTROL WORD FOR PRINT
BUFFER ADDRESS RESET
STORE FORU BACK INTO BUFFER CONTROL
RESTORNE F7
INPUT CARD SECTION
RESET CARD READ CONTROL WORD
STORE IN CARD READ CONTROL
RESTORNE F4, F5 AND F7
SUBROUTINE TO READ NEXT CHARACTER FROM A CARD

CONTROL WORD
TOO MANY CHARACTERS READ
CURRENT WORD OF 8 CHARACTERS

EXTRACT QUARTER WORD

RIGHT HAND QUARTER
SHIFT OVER FROM LEFT

EXTRACT 6 BITS
STORE CONTROL WORD BACK
RESTORE F5 AND F7

RESET LEFT/RIGHT COUNT
QUARTER WORD COUNT BRANCH IF NOT FOURTH
RESET TO FIRST QUARTER
AND INCREMENT ADDRESS

X FIELD
SET COUNT TO ONE
FAKE A COMMA READ
AND GO TO S FIELD TYPE

A OR C FIELD
COUNT TO M10
NUMBER OF CHARACTERS TO M9
SAVE NEXT CHARACTER IN M14
SAVE F7
FORMAT CONTROL IS SAVED
AND REPLACED SO THAT THE HOLLERITH PROGRAM CAN BE USED
RESTORE I/O CONTROL IN M15
CALL BINC
FIL
REBA JZM 6.BA2
CAM 12.2-2
CAM 14.4-4
CAM 15.M5
SFR 7.FRCNT
LFR 7.FCOM5
CAM 12.M9-9
JNM 12.BA1
CAM 9,8
BA1 CAM 7,4096
TRA BH
BA2 CAM 3.REBA
JPM 4.BE9
ATN 1,1,0
LFR 5
TRA BA4
BM ADM 2.2-2
BD ADM 2.4097
BQ ADM 2,4096
BL CAM 10.M9-1
CAM 9
JSB 0.BSUB
FIL
CAM 0.M2
ADM 8,11
SFR 6.FCOM5
CRN 15.10
CAM 11
CAM 2,M9
CAM 9
CAM 12
FIL
BD3 JZM 6,BD2

MOVE UP TO WORD BOUNFRARY

CONSTRUCT FAKE FORMAT CONTROL WORD

RESTORE F7

BRANCH IF LESS THAN 8 CHARACTERS LEFT
SET COUNT TO 8 CHARACTERS
SET MARKER IN M7 SO THAT CORRECT EXIT IS MADE FR M HOLLERITH

FETCH NEXT CONTROL WORD

M FIELD
D FIELD
Q FIELD
L FIELD
CLEAR NUMBER REGISTER
READ NUMBER

M/D/Q/L CONTROL TO MO
SAVE NEXT DIGIT

INPUT OUTPUT
COUNT TO M2
CLEAR POINT CONTROL
CLEAR SCALE FACTOR

NO MORE WORDS IN IO CONTROL WORD
SFR 7, FCOM6
JMP 11, BQA4A
JNM 0, BQA1
CALL READEC
SIA 8
JZM 0, BL1
LFR 7, M5
JMP 11, BDOUT
ANN 4, 3
TRA BD4
CAM 12, M8
TRA BD5
CAM 13, M8
TRA BD5
CAM 14, M8
TRA BD5
CAM 15, M8
BQ4
BD4
CAM 12, M8
TRA BD5
CAM 13, M8
TRA BD5
CAM 14, M8
TRA BD5
CAM 15, M8
SAVE F7
BRANCH IF OUTPUT
BRANCH IF OCTAL
READ DECIMAL NUMBER
STORE AS INTEGER IN M8
BRANCH IF L FIELD
DATA WORD TO M8
BRANCHC IF OUTPUT
FIRST QUARTER
SECOND QUARTER
THIRD QUARTER
FOURTH QUARTER
STORE BACK IN MEMORY
INC Q
NOT WORD BOUNDARY
INC WORD COUNT
DEC ITEM COUNT
RESTORE F7
DEC DECREASE REPEAT COUNT
REPEAT FIELD
RESTORE F6
NEXT CHARACTER IN FORMAT TO
SET LINK SHOULD REPEAT ENTRY ARISE
LAST CONTROL WORD USED
LOAD NEXT CONTROL WORD TO F5
BQ4A  TRA  BD3
    JZM  0,BM1
    CJU  0,BQ4
    CAM  0,-1
BM1   CAM  7,M5
    CAM  10
    TRA  BOUT1
BOUT  ORB  4,,
    CAM  7,M12
BOUT1 JNM  0,BOUT2
    CAD  9,3,M7
    CAM  12
    CAM  9
    CAM  7
    CAM  15
    CALL  FDP
    JUM  0,BD11
    TRA  BD7
BOUT2 CAM  9,M2-6
    CAM  8
    TRA  BOUT3
    CALL  FDBF
    ADM  9,-1
BOUT3 JPM  9,BOUT4
    CAM  11,-5
    CRM  7,12
    ANN  7,1
    CAM  8
    TRA  BOUT5
BOUT4 CALL  FLDBF
    ADM  9,-1
BOUT5 CJU  9,BOUT6
    JUM  8,BOUT5Z
    CAM  8,10

Branch if M field and if Q field
Reset switch for M field
Item to M
Clear repeat count
Extract quarter word for printing
Octal print
Number to accumulator
Store entry parameters for integer print
Print
Not an address print
Octal print start
Blank
Output character
Decrease count
Repeat character
Count for k octal digits
Extract first digit
Extract 2nd thru 5th digits
Count number of characters
Not zero
Zero code convert
BOUT5Z CALL FLDBF
CAM 9,1
OUTPUT CHARACTER

BOUT6 CJU 11,BOUT7
CJU 0,BD11
COUNT LENGTH OF FIELD
NOT ADDRESS PRINT

TRA BD7
BQ1 CSM 9,M2
CAM 7
OCTAL READ PROGRAM
CLEAR ASSEMBLY AREA

BQ2 CALL FRDBF
JZM 8,BQ3
READ CHARACTER
IGNORE BLANK

CRM 7,10
SHIFT ASSEMBLED WORD LEFT 3
ADM 7,M8
AND INSERT NEW OCTAL CHARACTER
ADM 8,-10
CODE CORRECT IF ZERO
JUM 8,BQ3
NOT ZERO
ADM 7,10

BQ3 CJU 9,BQ2
CAM 8,M7
COUNT CHARACTERS
ASSEMBLED QUARTER TO M8

CJU 0,BQ4
NOT ADDRESS READ
BL1 CAM 10
CLEAR REPEAT COUNT
CAM 5,M8
SET IN IO LIST IMAGE
SFR 7,FCOM6
TRA BD7
FIL
FCNTSC DECQ -80,INPBF,-2,M
FCNTS DECQ -133,OUTBF,-2,M
FZERO DECQ ,,
OUTBF BSS 17
INPBF BSS 10
PUSHDN BSS 5
ASSIGN FCOM1,FCOM2,FCOM3,FCOM4
ASSIGN FCNTS,FCOM12,FCOM13
ERROR CAM 1,ERM1
TRA ERPR
ERRORT CAM 1,ERM2
TRA ERPR
FERRPR CAM 1,ERM3
TRAM  ERPR
FRMERR CAM 1,ERM5
LFR 7,FRCNT
SFR 7,EMES+1
ERPR LFR 7,FCOM1
SFR 7,EMES
CAM 12
SFR 7,FCOM1
CALL PRINT
LFR 4,EMES
SFR 4,FCOM1
CALL SYSERR
FIL
EMES BSS 2
ERM1 DECO 3,EMES,1,MES1
ERM2 DECO 3,EMES,1,MES2
ERM3 DECO 3,EMES,1,MES3
ERM5 DECO 3,EMES+1,1,MES5
MES1 CHR 26H DATA CHARACTER INCORRECT Q5#
MES2 CHR 17H FIELD TOO SMALL Q5#
MES3 CHR 29H TOO MANY CHARACTERS ON LINE Q5#
MES5 CHR 14H FORMAT ERROR Q5#
ASSIGN FCOM7,FCOM8,FCOM9
ASSIGN FCOM6,FCOM5,FCOM24
ASSIGN FBFWD,FT5,FT6
WRITE EQU 512
8.4 Program Descriptions

A number of subroutines are in preparation for the card operating system and should be available by June. (No descriptions are yet available.)

They will include:

Logarithm
Sin/Cos
Square Root
Exponential
Integration of Ordinary Differential Equations
Solution of Linear Equations
Roots of Polynomial
Quadrature