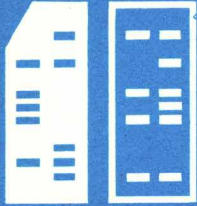


ILLIAC II MANUAL
USE OF THE NEW ILLINOIS COMPUTER INSTALLATION

Edited by
C. W. Gear

March 1963



DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

ILLIAC II MANUAL
USE OF THE NEW ILLINOIS COMPUTER INSTALLATION

Edited by

C. W. Gear

March 1963

CONTENTS

Chapter	Prepared by
1. INTRODUCTION	C. W. Gear
2. THE OPERATING SYSTEM	
3. A MACHINE DESCRIPTION AND THE MACHINE LANGUAGE	D. B. Gillies
4. NICAP, THE ASSEMBLY PROGRAM	C. W. Gear
5. SYSTEM INPUT-OUTPUT AND AUXILIARY STORAGE	
6. AUXILIARY EQUIPMENT	
7. COMPILERS	C. D. Shepard
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.

Date:	3/5/63
Section:	1.1
Page:	1 of 2
Change:	

Additionally, the machine has the capabilities of performing 32 input/output operations simultaneous with compute 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.

Date:	3/5/63
Section:	1.1
Page:	2 of 2
Change:	

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.

Date:	3/5/63
Section:	1.2
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	1.3
Page:	1 of 3
Change:	

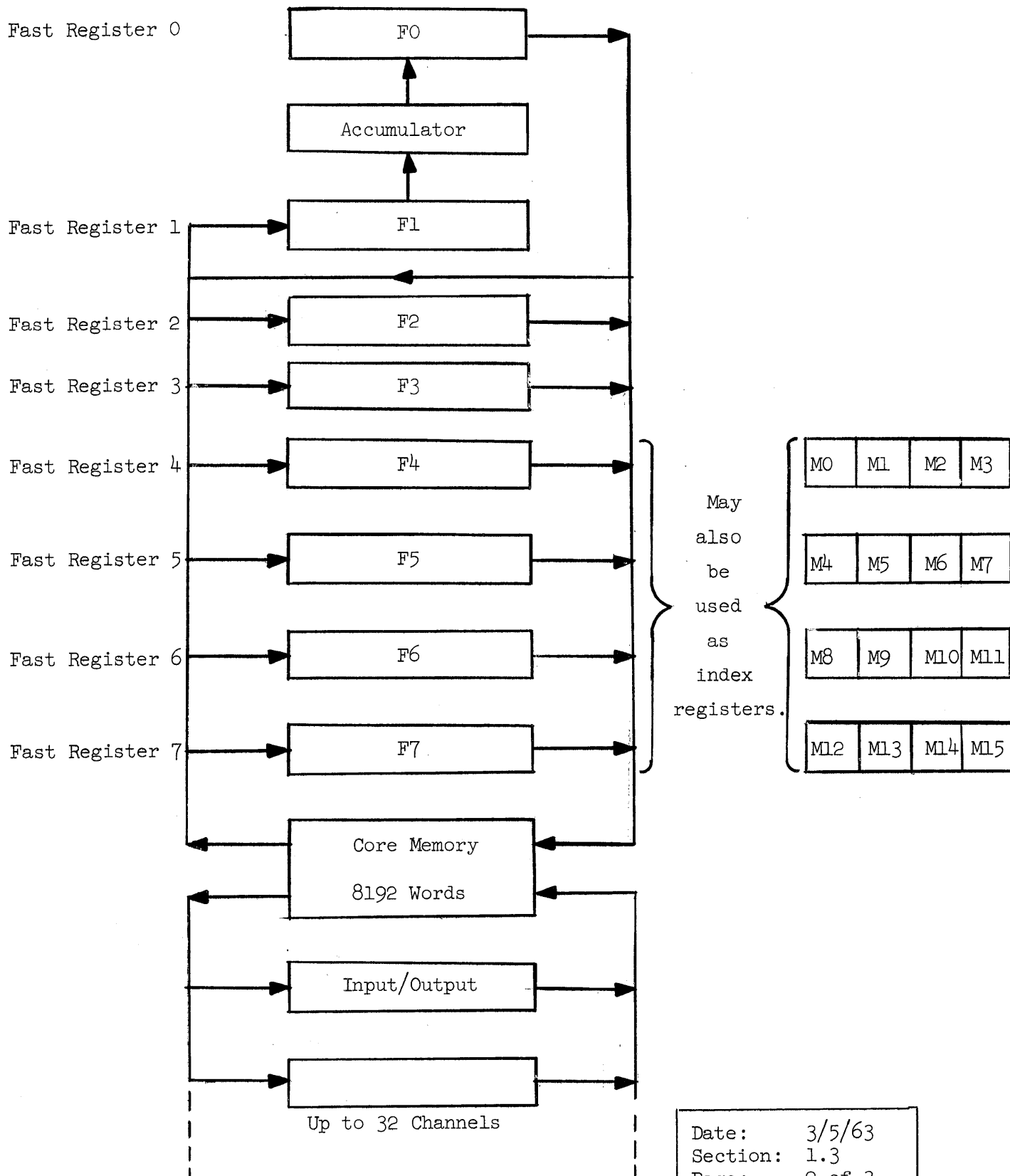


Figure 1

Date:	3/5/63
Section:	1.3
Page:	2 of 3
Change:	

- (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 FO, 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.

Date:	3/5/63
Section:	1.3
Page:	3 of 3
Change:	

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.

Date:	3/5/63
Section:	1.4
Page:	1 of 1
Change:	

CHAPTER 2. THE OPERATING SYSTEM

TABLE OF CONTENTS

	Change	Date
2.1 Introduction		7/8/64
2.2 The Core-Load Principle		7/8/64
2.2.1 System Translation and Relocation		7/8/64
2.2.2 Binary Cards		7/8/64
2.3 Batch Processing		7/8/64
2.3.1 Messages from the Batch Processing System		7/8/64
2.4 The System Library Tape		7/8/64
2.A.1 APPENDIX 1		7/8/64
2.A.2 APPENDIX 2		7/8/64

Date:	7/8/64
Section:	Chapter 2 Contents
Page:	1 of 1
Change:	

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.

Date:	7/8/64
Section:	2.1
Page:	1 of 1
Change:	

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.

Date:	7/8/64
Section:	2.2
Page:	1 of 1
Change:	

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.

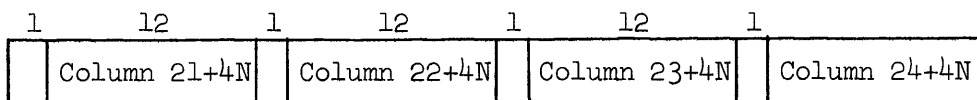
Date:	7/8/64
Section:	2.2.1
Page:	1 of 1
Change:	

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:

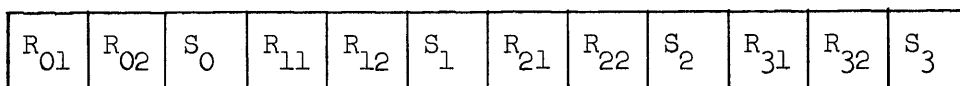
Column 1 7-9 punch for binary

Columns $10+N$, $21+4N$, $22+4N$, $23+4N$ and $24+4N$ for $N = 1, 2, \dots, 14$ represent 14 52-bit words W_1, W_2, \dots, 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 .



Word W_N

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

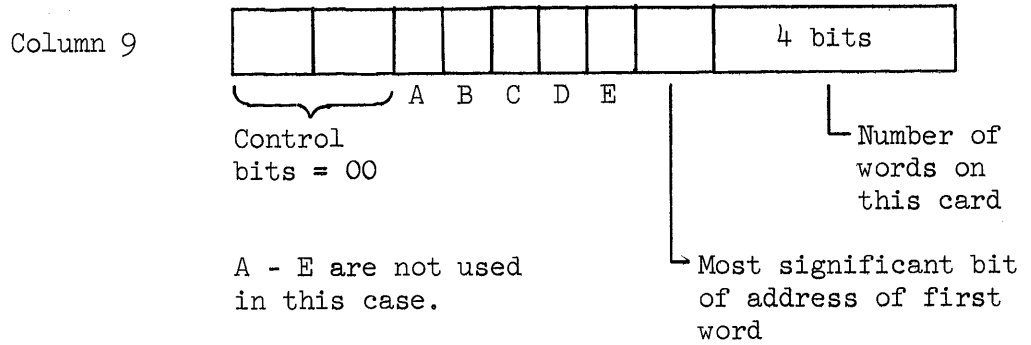


Column $10+N$ (12 bits)

S_i is the sign bit, $R_{i1}R_{i2}$ 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 i th quarter of word W_N .

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

Date:	7/8/64
Section:	2.2.2
Page:	1 of 3
Change:	



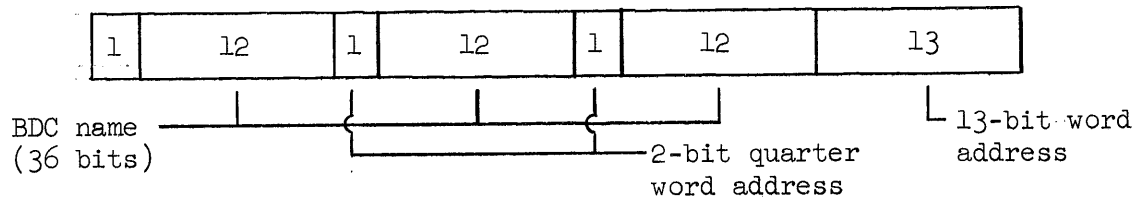
Column 10 contains the least significant 12 bits of the address of the first word to be loaded from this card.

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:



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 , ..., 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 sub-routines. 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.

Date:	7/8/64
Section:	2.2.2
Page:	3 of 3
Change:	

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.

Date:	7/8/64
Section:	2.3
Page:	1 of 2
Change:	

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.

Date:	7/8/64
Section:	2.3
Page:	2 of 2
Change:	

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 1 in the process. Also, the location at which the CALL SYSERR was executed will be printed.

Date:	7/8/64
Section:	2.3.1
Page:	1 of 1
Change:	

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.

Date:	7/8/64
Section:	2.4
Page:	1 of 1
Change:	.

2.A.1 APPENDIX 1
SYSTEM CONTROL CARDS

<u>Message</u>	<u>Comments</u>
GO	Must appear before any programs.
DUMP	Causes a memory dump only if SYSERR is called.
PRINT OBJECT	Should be used with assembly if listing desired.
PUNCH OBJECT	
NICAP	
BINARY	No NICAP or compiler program may follow this.
DATA	Last card read by system. Not necessary if no data.

Date:	7/8/64
Section:	2.A.1
Page:	1 of 1
Change	

2.A.2 APPENDIX 2

PROGRAMS ON LIBRARY TAPE

<u>Major Name</u>	<u>Minor Names</u>	<u>Subroutines Used</u>
ATAN1		
DVD1		
EXPL		
GQUL		
LAG6		
LGT1	LGB1, LGEL	
LGUN		
PRINT	PUNCH, READ	SYSIØ, SYSERR
RKGL		
SIN1	CØS1	
SQRL		
SYSAUX		
SYSERR		
SYSIØ		
SYSTEM		

Date:	7/8/64
Section:	2.A.2
Page:	1 of 1
Change	

CHAPTER 3. A MACHINE DESCRIPTION AND THE MACHINE LANGUAGE

TABLE OF CONTENTS

	Change	Date
3.1 Introduction		
3.1.1 Machine Features		3/5/63
3.1.2 Additional Equipment		3/5/63
3.2 General Mode of Operation		3/5/63
3.2.1 Delayed Control		3/5/63
3.2.2 Advanced Control		3/5/63
3.2.3 Interplay		3/5/63
3.2.4 Use of Immediate Access Memory		3/5/63
3.2.5 Use of Fast Memory		3/5/63
3.2.6 Program Interrupt		3/5/63
3.3 Order Code for Floating-Point Arithmetic		3/5/63
3.3.1 The Floating-Point Accumulator		3/5/63
3.3.2 Zero and Overflow		3/5/63
3.3.3 Normalization		3/5/63
3.3.4 Addition and Subtraction		3/5/63
3.3.5 Multiplication		3/5/63
3.3.6 Division		3/5/63
3.3.7 Round-Off		3/5/63
3.3.8 Correct Overflow and Detect Zero		3/5/63
3.3.9 Floating-Point Orders	1	7/9/64
3.4 Orders Which Do Not Involve Floating Point		3/5/63
3.4.1 Interplay Orders		3/5/63
3.4.2 Block Reservation Orders		3/5/63
3.4.3 Advanced Control Orders	1	7/9/64
3.4.4 Modifier Arithmetic		3/5/63
3.5 Tables		
3.5.1 Table 1. Address Construction		3/5/63
3.5.2 Table 2. Special Case Information on Instructions		3/5/63
3.5.3 Table 3. Order Code Index		3/5/63
3.5.4 Table 4. Order Code Listed Numerically		3/5/63
3.5.5 Table 5. Additional Mnemonics	1	7/9/64

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:

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

Date:	3/5/63
Section:	3.1.1
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	3.1.2
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	3.2
Page:	1 of 2
Change:	

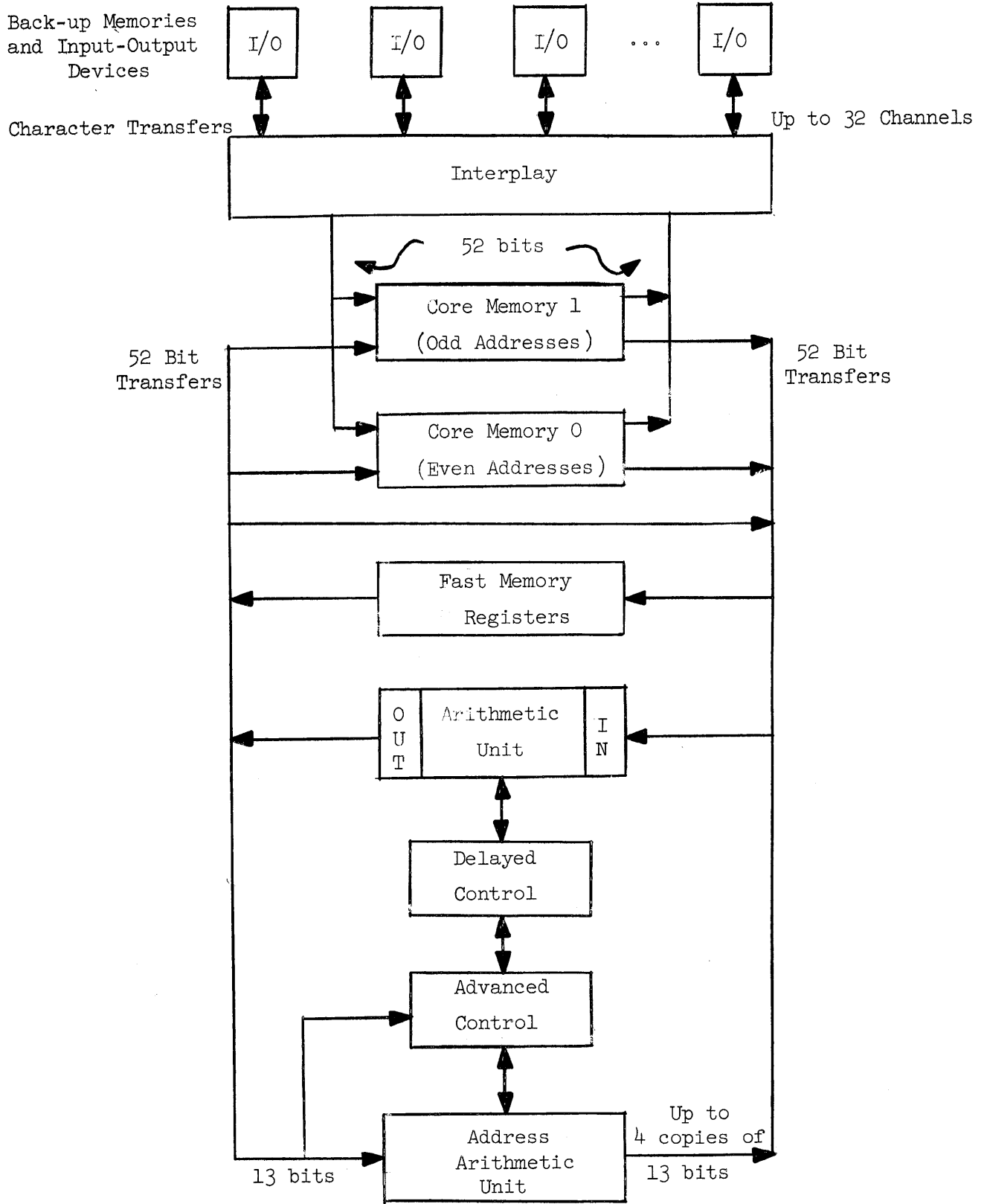


Figure 1
Data Paths for ILLIAC II

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.

Date:	3/5/63
Section:	3.2.1
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	3.2.2
Page:	1 of 1
Change:	

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,

block 0 comprising locations	0 to 255
block 1 comprising locations	256 to 511
:	:
:	:
:	:
block 31 comprising locations	7936 to 8191

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.

Date:	3/5/63
Section:	3.2.3
Page:	1 of 2
Change:	

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).

Date:	3/5/63
Section:	3.2.3
Page:	2 of 2
Change:	

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, 409~~6~~⁷ 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).

Date:	3/5/63
Section:	3.2.4
Page:	1 of 1
Change:	

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

Date:	3/5/63
Section:	3.2.5
Page:	1 of 3
Change:	

instruction STR F6 would normalize, round off and store the contents of the floating-point accumulator into register F6, thereby overwriting modifiers M8, M9, M10, M11. In the latter case M8 will have most significant 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

Date:	3/5/63
Section:	3.2.5
Page:	2 of 3
Change:	

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~~^{LFR} can load F2, ..., F7 but not F0 or F1.
- (4) SFR can store F0, F2, ..., F7 but not F1.

Date:	3/5/63
Section:	3.2.5
Page:	3 of 3
Change:	

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.

Date:	3/5/63
Section:	3.2.6
Page:	1 of 2
Change:	

- (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.

Date:	3/5/63
Section:	3.2.6
Page:	2 of 2
Change:	

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 SIA: 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 $N + (M_B) \bmod 8192$.

If $C = 3$ and $B < 8$ the operand is contained in fast register F_B .

Date:	3/5/63
Section:	3.3
Page:	1 of 3
Change	

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 $F1$, so $C = 3$, $B = 1$ is illegal here. For $C = 3$, $B \geq 9$ an operand destination is meaningless.

Date:	3/5/63
Section:	3.3
Page:	2 of 3
Change:	

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 $C = 3$, $B \neq 8$ 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.

Date:	3/5/63
Section:	3.3
Page:	3 of 3
Change	

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, \dots, a_{88} in twos complement notation, with value

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

The first 45 bits of A (a_0, \dots, 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 ($0, a_{45}, \dots, a_{88}$) 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, \dots, e_0 with integer value

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

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.

Date:	3/5/63
Section:	3.3.1
Page:	1 of 1
Change:	

3.3.2 Zero and Overflow

The representation in 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.

Date:	3/5/63
Section:	3.3.2
Page:	1 of 2
Change:	

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 $0 \cdot 4^{-64}$ (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.

Date:	3/5/63
Section:	3.3.2
Page:	2 of 2
Change:	

3.3.3 Normalization

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

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

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.

Date:	3/5/63
Section:	3.3.3
Page:	1 of 1
Change:	

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:

Case 1	$e - y \leq 45$	obey Clear Add	six steps
Case 2	$-44 \leq e - y \leq 0$	about $5 + e - y $	steps
Case 3	$1 \leq e - y \leq 22$	about $5 + 2 e - y $	steps
Case 4	$23 \leq e - y \leq 44$	about $-11 + e - y $	steps
Case 5	$45 \leq e - y$	bypass	three steps

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

Date:	3/5/63
Section:	3.3.4
Page:	1 of 1
Change:	

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 $a_\ell = 0$ normalization is not necessary (and is not done), since the product $(a \cdot x) 4^{e+y}$ is exact. Likewise if a_ℓ 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 $a_\ell = 0$ 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).

Date:	3/5/63
Section:	3.3.5
Page:	1 of 1
Change:	

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_0 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.

Date:	3/5/63
Section:	3.3.6
Page:	1 of 1
Change:	

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:

$$\begin{aligned} a_r &= a_m && \text{if } a_\ell < \frac{1}{2} \\ a_r &= a_m + 4^{-22} a_{44} && \text{if } a_\ell = \frac{1}{2} * \\ a_r &= a + 4^{-22} && \text{if } a_\ell > \frac{1}{2} \end{aligned}$$

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.

Date:	3/5/63
Section:	3.3.7
Page:	1 of 1
Change:	

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."

Date:	3/5/63
Section:	3.3.8
Page:	1 of 1
Change:	

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

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

NOT (104) Clear Add Digitwise Complement. Clear Z and a_0 . 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 $C = 3$, $B < 8$ (Also named LFl) and $B \neq 1$, a_m is not changed. Otherwise a_m is replaced logically by $2a_m \bmod 2$. 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

MPY (120) 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.

DIV (121) 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 $\left(\frac{a}{4x}\right)$ rounded in a_m , set $a_\ell = 0$ 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.

NDV (122) Negative Divide. Identical to DIV except that the divisor is $(-x) \cdot 4^y$.

VID (123) Inverse Divide. The accumulator is normalized and rounded, and a_r and x are interchanged and a_ℓ 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 $\left(\frac{x}{a_r}\right) \cdot 4^{y-e}$ or $\left(\frac{x}{4a_r}\right) \cdot 4^{y-e+1}$ in a manner analogous to that described under DIV above.

Date:	7/9/64
Section:	3.3.9
Page:	3 of 9
Change:	1

- LAL (141) Load A Least. Z is cleared and the digits a_{45}, \dots, a_{88} are set equal to the non-sign digits of x, namely $x_1x_2\dots x_{44}$. 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 ≤ -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.

Date:	7/9/64
Section:	3.3.9
Page:	4 of 9
Change:	1

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_l 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. a_r is formed. 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.

Date:	7/9/64
Section:	3.3.9
Page:	5 of 9
Change:	1

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 ≥ 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 $a_r = a_m + 2^{-44}\epsilon$ where $\epsilon = 0$ or 1 . Then $a \cdot 4^e = (a_m + 4^{-22}a_\ell) \cdot 4^e = [a_r + 4^{-22}(a_\ell - \epsilon)] \cdot 4^e = a_r \cdot 4^e + (a_\ell - \epsilon)4^{e-22}$. Now STR stores a number numerically equal to $a_r \cdot 4^e$. STC after doing this, transfers $a_\ell - \epsilon$ to a_m , clears a_ℓ to zero and subtracts 22 from e so the accumulator holds the remainder from the store operation. Z is set if $a_\ell - \epsilon = 0$ 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.

Date:	7/9/64
Section:	3.3.9
Page:	6 of 9
Change:	1

- 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_l . 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 $a_m : 4^{22}$ 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. $a_l \cdot 4^e$ 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.

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

For the next group of orders, Advanced Control places the same address into all four quarters of F1. This address is interpreted modulo 256 for the first four and modulo 128 for SRS, LRS, and is called y^* 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_l 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 F0, then the 1/4 word of F0 aligned with

Date:	7/9/64
Section:	3.3.9
Page:	8 of 9
Change:	1

M_B is copied into M_B . If the modifier register specified is M3, M7, M11, or M15, the last 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 M_B . Otherwise a different quarter word is copied (see SEX). The C field of the SIA order has no effect.

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

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 F1, 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 F1 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.

Date:	3/5/63
Section:	3.4
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	3.4.1
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	3.4.2
Page:	1 of 1
Change:	

3.4.3 Advanced Control Orders

LIN (060) Load IN Register. Four copies of the constructed address are placed in Fl. 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

ATN 2,0
ATN 5,0
CAD 7,0 or 3 short orders.

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.

Date:	7/9/64
Section:	3.4.3
Page:	1 of 6
Change:	1

ORB (061) Logical OR with B Digits of the Next Instruction. The right-most 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.

Date:	7/9/64
Section:	3.4.3
Page:	2 of 6
Change:	1

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, 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 instruc-

tion, 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.

Date:	7/9/64
Section:	3.4.3
Page:	3 of 6
Change:	1

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, CJS, 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.

Date:	7/9/64
Section:	3.4.3
Page:	4 of 6
Change:	1

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 JLH 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 $a \geq 0$).
 - 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
 - B = 11 OV not on
 - B = 12 *Green Sw. Center*
 - B = 13 *Green Sw. Down (normal)*
 - B = 14 Digit $a_0 = 0$ (useful mainly in logical operations.
 - B = 15 Digit $a_0 = 1$ (useful mainly in logical operations.
- } and then clear OV if it was on.

Note that if B \neq 0 or 1 this jump is conditional on the arithmetic result after Delayed Control

Date:	7/9/64
Section:	3.4.3
Page:	5 of 6
Change:	1

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.

Date:	7/9/64
Section:	3.4.3
Page:	6 of 6
Change:	1

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 O 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.

Date:	3/5/63
Section:	3.4.4
Page:	1 of 3
Change:	

- 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 $0 \leq p \leq 15$. 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 are 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.

Date:	3/5/63
Section:	3.4.4
Page:	2 of 3
Change:	

- EQM (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 is 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.

Date:	3/5/63
Section:	3.4.4
Page:	3 of 3
Change:	

3.5 Tables

3.5.1 Table 1. Address Construction

Normal: LAL, CAD, CSB, 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

Normal, C = 3, B ≠ 8 illegal, address used as operand: CAE, CSE, ADE,
SBE, SRS, LRS,
LIN, *PID,
*POD, *IBT,
*BBF, *FBB,
JLH, ATN, SFN,
ORB

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, JSB, 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.

Date:	3/5/63
Section:	3.5.1
Page:	1 of 1
Change:	

3.5.2 Table 2. Special Case Information on Instructions

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

Date:	3/5/63
Section:	3.5.2
Page:	1 of 1
Change:	

3.5.3 Table 3. Order Code Index

<u>Order Codes</u>	<u>Section No.</u>	<u>Page No.</u>	<u>Order Codes</u>	<u>Section No.</u>	<u>Page No.</u>
ADD 112	3.3.9	2	LAL 141	3.3.9	4
ADE 113	3.3.9	8	LDM 071	3.4.3	6
ADM 067	3.4.4	1	LFR 070	3.4.3	6
AND 105	3.3.9	1	LIN 060	3.4.3	1
ANM 047	3.4.4	2	LOR 106	3.3.9	1
ASC 116	3.3.9	6	LRS 145	3.3.9	8
*ASN 032	3.4.3	2	MPY 120	3.3.9	3
ATN 021	3.4.3	1	NAM 045	3.4.4	3
*BBF 043	3.4.2	1	NDV 122	3.3.9	3
BLS 107	3.3.9	2	NOM 044	3.4.4	3
CAD 102	3.3.9	1	NOT 104	3.3.9	2
CAE 117	3.3.9	8	ORB 061	3.4.3	2
CAM 027	3.4.4	1	ORM 046	3.4.4	2
CAT 103	3.3.9	1	*PID 023	3.4.1	1
CJF 057	3.4.3	3	*POD 062	3.4.1	1
CJS 055	3.4.3	3	SAL 136	3.3.9	7
CJU 077	3.4.3	4	SAM 135	3.3.9	7
CJZ 037	3.4.3	4	SBE 111	3.3.9	8
CNM 024	3.4.4	2	SBM 065	3.4.4	1
CRM 026	3.4.4	2	SEQ 132	3.3.9	7
CSB 100	3.3.9	1	SEX 137	3.3.9	8
CSE 115	3.3.9	8	SFN 041	3.4.3	1
CSM 025	3.4.4	1	SFR 030	3.4.3	6
CST 101	3.3.9	1	SIA 133	3.3.9	9
DAV 142	3.3.9	4	SIF 131	3.3.9	7
DIV 121	3.3.9	3	SRM 143	3.3.9	6
EOM 066	3.4.4	2	SRS 147	3.3.9	8
EQM 064	3.4.4	3	SSC 114	3.3.9	6
*F'BF 042	3.4.2	1	*SSN 072	3.4.3	2
*IBT 022	3.4.1	1	*SSR 073	3.4.3	2
JDC 056	3.4.3	5	STC 126	3.3.9	6
JLH 054	3.4.3	1	STF 130	3.3.9	5
JNM 034	3.4.3	4	STN 127	3.3.9	5
JPM 074	3.4.3	4	STR 124	3.3.9	4
JSB 076	3.4.3	5	STU 134	3.3.9	5
JUM 075	3.4.3	4	SUB 110	3.3.9	2
JZM 035	3.4.3	4	VID 123	3.3.9	3
			XCH 125	3.3.9	5

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:	

3.5.4 Table 4. Order Code Listed Numerically

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

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.

Date:	3/5/63
Section:	3.5.4
Page:	1 of 1
Change:	

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

Date:	7/9/64
Section:	3.5.5
Page:	1 of 1
Change:	1

CHAPTER 4. NICAP, THE ASSEMBLY PROGRAM

TABLE OF CONTENTS

	Change	Date
4.1 Introduction		3/5/63
4.2 Card Format		3/5/63
4.2.1 Location Field	1	7/9/64
4.2.2 Mnemonic Field		3/5/63
4.2.3 Address Fields		3/5/63
4.2.4 Comments Field	1	7/9/64
4.2.5 Identification Field		3/5/63
4.3 Address Construction		3/5/63
4.3.1 Machine Evaluation of Address Expressions		3/5/63
4.3.2 Illegal Use of Names in Address Fields	1	7/9/64
4.4 Orders		
4.4.1 Type 1 Orders		3/5/63
4.4.2 Type 2 Orders		3/5/63
4.4.3 Type 3 Orders		3/5/63
4.4.4 Type 4 Orders		3/5/63
4.4.5 Type 5 Orders		3/5/63
4.4.6 Type 6 Orders		3/5/63
4.4.7 Type 7 Orders		3/5/63
Type 7A Orders		
4.4.8 Type 8 Orders	1	7/9/64
4.4.9 Type 9 Orders		3/5/63
4.4.10 Type 10 Orders		3/5/63
4.4.11 Type 11 Orders		3/5/63
4.4.12 Type 12 Order	1	7/9/64
4.4.13 Type 13 Orders	1	7/9/64
4.4.14 Illegal Orders		7/9/64
4.5 Pseudo Orders		3/5/63
4.5.1 Directives	2	7/9/64
4.5.2 Data-Loading Pseudo Orders	1	7/9/64
4.6 Input/Output Pseudo Instructions		3/5/63
4.7 Macro Orders		3/5/63
4.8 Notes on Simple Programming in Assembler Language	1	7/9/64
4.9 Program Listing	2	7/9/64
4.10 Tables		
4.10.1 Table 1. Order Code Index	1	7/9/64
4.10.2 Table 2. Pseudo Orders	2	7/9/64
4.10.3 Table 3. BCD Tape and Card Code	1	11/8/63

Date:	7/9/64
Section:	Chapter 4 Contents
Page:	1 of 1
Change:	1

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.

Date:	3/5/63
Section:	4.1
Page:	1 of 1
Change:	

4.2 Card Format

1	678	13	14	15	32 33	72 73 80
---	-----	----	----	----	-------	----------

Location Mnemonic Address

Comment

Identification

Date: 3/5/63
Section: 4.2
Page: 1 of 1
Change:

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 MO, M1, ..., M15, and FO, 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.

Date:	7/9/64
Section:	4.2.1
Page:	1 of 1
Change:	1

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.

Date:	3/5/63
Section:	4.2.2
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.2.3
Page:	1 of 1
Change:	

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.

Date:	7/9/64
Section:	4.2.4
Page:	1 of 1
Change:	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.

Date:	3/5/63
Section:	4.2.5
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.3
Page:	1 of 1
Change:	

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 F1 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^{44} . 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

$$7/2 + 9/2$$

will give an address of

$$3 + 4 = 7,$$

whereas

$$M3/2 + M4/2$$

will give an address of

Date:	3/5/63
Section:	4.3.1
Page:	1 of 2
Change:	

$$3 \frac{1}{2} + 4 \frac{1}{2} = 8$$

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.

Date:	3/5/63
Section:	4.3.1
Page:	2 of 2
Change:	

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.

Date:	7/9/64
Section:	4.3.2
Page:	1 of 1
Change:	1

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

CAD 8,3,A

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

CAD A

Date:	3/5/63
Section:	4.4
Page:	1 of 1
Change:	

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 F5, then the order uses the contents of that fast register as the operand. It is assembled as one short order, e.g., CAD F5 has a B field of 5 and a C field of 3, so is equivalent to CAD 5,3, in the normal form and puts the contents of F5 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:

CAD M5	is equivalent to	CAD 5,0,
CAD M5 + 301	is equivalent to	CAD 5,2,301
CAD M4 + M7	is equivalent to	ATN 4,0, 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 and B ≠ 8, 9, or 10

B may also be a tag if C ≠ 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.

Date:	3/5/63
Section:	4.4.1
Page:	2 of 2
Change:	

4.4.2 Type 2 Orders

ASC	ADD to Store and Clear
SAL	Store A Least
SAM	Store A Most
SEQ	Store with Exponent Equal
SIF	Store Integer Part in Floating Point
SRM	Store Remainder
SSC	Subtract, Store and Clear
STC	Store and Clear
STF	Store Fixed Point Rounded
STN	Store Negatively
STR	Store Rounded and Normalized
STU	Store Unnormalized but Rounded
XCH	Exchange

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, F1 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 $C = 3$.

Date:	3/5/63
Section:	4.4.2
Page:	1 of 1
Change:	

4.4.3 Type 3 Orders

ADE	Add to Exponent
ATN	Add to Next
*BBF	Busy Block Flipflop
CAE	Clear Add Exponent
CSE	Clear Subtract Exponent
*FBF	Free Block Flipflop
*IBT	Initiate Block Transfer
JLH	Jump to Left-hand Side
LIN	Load In Register
LRS	Long Right Shift
ORB	OR to B Digits of Next
*PID	Prepare Input Device
*POD	Prepare Output Device
SBE	Subtract from Exponent
SFN	Subtract from Next
SRS	Short Right Shift

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.

Date:	3/5/63
Section:	4.4.3
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.4.4
Page:	2 of 2
Change:	

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.

Date:	3/5/63
Section:	4.4.5
Page:	1 of 1
Change:	

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 CJZ.

4.4.6 Type 6 Orders

CJF	Count and Jump to First (if nonzero)
CJS	Count and Jump to Second (if nonzero)

The normal form of addressing

CJS B,C,

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

CJS 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 CAJ (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.)

Date:	3/5/63
Section:	4.4.6
Page:	1 of 1
Change:	

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, COS+7

to jump to the $(4 \times 7 + 1) = 29$ th quarter word after the start of the subroutine COS. In fact this will work even if the subroutine were not to

Date:	3/5/63
Section:	4.4.7
Page:	1 of 3
Change:	

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,

CJU4,2,COS+7

is equivalent to

CJU4,1,108

However, beware:

CJU4,2,7+COS

is equivalent to

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:

JSB3,COS

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

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).

Date:	3/5/63
Section:	4.4.7
Page:	2 of 3
Change:	

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.

Date:	3/5/63
Section:	4.4.7
Page:	3 of 3
Change:	

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,M13

assembles as

ATN13,0,

LFR5,0,

NOTE: SFR may not use F1.

Date:	3/5/63
Section:	4.4.9
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.4.10
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.4.11
Page:	1 of 1
Change:	

4.4.12 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 $B \neq 1$. If no address is used in BLS, it assembles as BLS 1,3,. 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.

Date:	7/9/64
Section:	4.4.12
Page:	1 of 1
Change:	1

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.

Date:	7/9/64
Section:	4.4.13
Page:	1 of 1
Change:	1

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.

Date:	7/9/64
Section:	4.4.14
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.5
Page:	1 of 1
Change:	

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

FLD 4

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

FLD

advances it to the next even word boundary.

Date:	7/9/64
Section:	4.5.1
Page:	1 of 5
Change:	2

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 0 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.,

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.

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

The following six pseudo orders do not affect the instruction counter.

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

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.

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

EQU (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.

Date:	7/9/64
Section:	4.5.1
Page:	4 of 5
Change:	2

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.

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

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

1 101 000 111 001	0 000 000 011 010
1 5 0 7 1	3 2

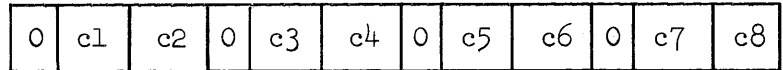
OCTQ and DECQ cause outside names to be defined as labels.

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

Date:	7/9/64
Section:	4.5.2
Page:	1 of 3
Change:	1

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:



DEC

first performs a FILL, 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:

<u>Punched as</u>	<u>Value</u>
72	$+.74 \times 10^2$
- 27.1	$-.271 \times 10^2$
+30.E-02	$+.301 \times 10^1$
7.5E06	$+.75 \times 10^7$
- 3.32-05	$-.332 \times 10^{-4}$
-3.32E-5	$-.332 \times 10^{-4}$

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

Date:	7/9/64
Section:	4/5/2
Page:	2 of 3
Change:	1

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.

Date:	7/9/64
Section:	4.5.2
Page:	3 of 3
Change:	1

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 FILL, 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.

Date:	3/5/63
Section:	4.6
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	4.7
Page:	1 of 1
Change:	

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 $p(x) = A_N + A_{N-1}x + \dots + A_0x^N$ where the coefficients A_0, A_1, \dots, A_N are in locations $A, A+1, \dots, A+N-1$ and x is in location X .

Date:	7/9/64
Section:	4.8
Page:	1 of 3
Change:	1

The program:

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

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 FILL.

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
        LFR 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.

Date:	7/9/64
Section:	4.8
Page:	2 of 3
Change:	1

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,S1      Load four modifiers
                                CSM 8,N
                                FIL
L1   CAD 4,1,
      ADD 5,1,
      ADD 6,1,
      STR 7,1,
      CAJ 8,L1
      ...
      ...
      ...
      ...
      FIL
S1   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.

Date:	7/9/64
Section:	4.8
Page:	3 of 3
Change:	1

4.9 Program Listing

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.

Date:	7/9/64
Section:	4.9
Page:	1 of 1
Change:	2

4.10 Tables4.10.1 Table 1. Order Code Index

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

Date:	7/9/64
Section:	4.10.1
Page:	1 of 2
Change:	1

4.10.1 Table 1. Order Code Index (Continued)

<u>Order</u>	<u>Type</u>	<u>Section No.</u>	<u>Page No.</u>	<u>Order</u>	<u>Type</u>	<u>Section No.</u>	<u>Page No.</u>
ORB	3	4.4.3	1	STF	2	4.4.2	1
ORM	4	4.4.4	1	STN	2	4.4.2	1
ORN	5	4.4.5	1	STR	2	4.4.2	1
*PID	3	4.4.3	1	STU	2	4.4.2	1
*POD	3	4.4.3	1	SUB	2	4.4.1	1
SAL	2	4.4.2	1	TEI	8	4.4.8	1
SAM	2	4.4.2	1	TLN	8	4.4.8	1
SBE	3	4.4.3	1	TLP	8	4.4.8	1
SBM	4	4.4.4	1	TN	8	4.4.8	1
SBN	5	4.4.5	1	TNO	8	4.4.8	1
SEQ	2	4.4.2	1	TNOR	8	4.4.8	1
SEX	11	4.4.11	1	TO	8	4.4.8	1
SFN	3	4.4.3	1	TOR	8	4.4.8	1
SFR	9	4.4.9	1	TP	8	4.4.8	1
SIA	11	4.4.11	1	TRA	8	4.4.8	1
SIF	2	4.4.2	1	TU	8	4.4.8	1
SRM	2	4.4.2	1	TZ	8	4.4.8	1
SRS	3	4.4.3	1	TZN	8	4.4.8	1
SSC	2	4.4.2	1	TZP	8	4.4.8	1
*SSN	13	4.4.13	1	VID	1	4.4.1	1
*SSR	13	4.4.13	1	XCH	2	4.4.2	1
STC	2	4.4.2	1				

Date:	7/9/64
Section:	4.10.1
Page:	2 of 2
Change:	1

4.10.2 Table 2. Pseudo Orders

<u>Pseudo Order</u>	<u>Section No.</u>	<u>Page No.</u>
ASSIGN	4.5.1	2
BES	4.5.1	2
BSS	4.5.1	2
CALL	4.5.2	3
CHR	4.5.2	1
COMMON	4.5.1	4
DEC	4.5.2	2
DECQ	4.5.2	1
END	4.5.1	4
ENTRY	4.5.1	5
EQUF	4.5.1	3
EQL	4.5.1	3
EQU	4.5.1	3
EQU	4.5.1	3
EQU	4.5.1	3
ERASE	4.5.1	5
FIL	4.5.1	1
FLD	4.5.1	1
GO	4.5.1	2
MACRO	4.5.1	3
OCTQ	4.5.2	1
ORG	4.5.1	1

Date:	7/9/64
Section:	4.10,2
Page:	1 of 1
Change:	2

TABLE OF PERMISSIBLE CHARACTERS

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

† Character is not normally used. When it is used it will not be considered a sign.

Date:	11/8/63
Section:	4.10.3
Page:	1 of 1
Change:	1

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

CHAPTER 5. SYSTEM INPUT/OUTPUT AND AUXILIARY STORAGE

TABLE OF CONTENTS

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

Date:	7/10/64
Section:	Chapter 5
	Contents
Page:	1 of 1
Change:	1

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.

Date:	7/10/64
Section:	5.2
Page:	1 of 1
Change:	

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  
DECQ 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!)

Date:	7/10/64
Section:	5.2.1
Page:	1 of 3
Change:	

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

blank = BCD card

1 = line for printer

2 = 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.

Date:	7/10/64
Section:	5.2.1
Page:	3 of 3
Change:	

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

```
CALL SYSAUX
DECQ ØP, A, EØF, U
```

where ØP is the operation type

A is the starting core address

EØF is the EØF 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 SYSIØ, 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

Date:	7/10/64
Section:	5.2.2
Page:	1 of 2
Change:	

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 EØF 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 EØF Mark

Date:	7/10/64
Section:	5.2.2
Page:	2 of 2
Change:	

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.

Date:	11/8/63
Section:	5.3
Page:	1 of 1
Change:	

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.

Date:	11/8/63
Section:	5.3.1
Page:	1 of 1
Change:	

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.

Fw.d or Iw.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 . 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 suppressed, 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, $.23443 \times 10^2$. Since card readers and printers do not handle superscripts, the exponent is indicated by preceding it with an E. The general form of a number in an E field is

$$\pm .xxx\dots xE\pm ee$$

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

$$\pm . E\pm ee$$

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 location, so the corresponding entry in the I/O list must refer

Date:	7/10/64
Section:	5.3.1.1
Page:	2 of 6
Change:	1

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

Date:	7/10/64
Section:	5.3.1.1
Page:	3 of 6
Change:	1

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.

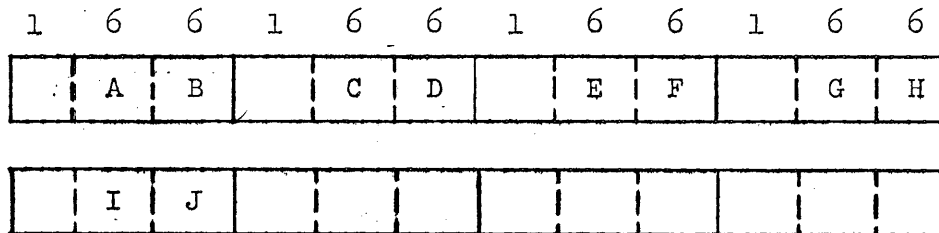
Sw Space. On output, Sw causes w spaces to be printed. On input, Sw causes w columns to be ignored.

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

ABCDEFGHIJ

then the two computer words involved would contain



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.

Date:	7/10/64
Section:	5.3.1.1
Page:	4 of 6
Change:	1

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.

Date:	7/10/64
Section:	5.3.1.1
Page:	5 of 6
Change:	1

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.

Mn

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.

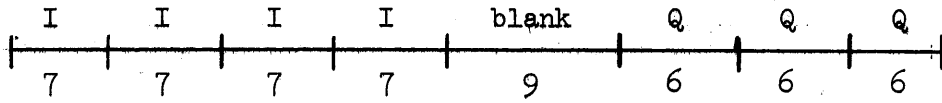
Double Precision

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.

Date:	7/10/64
Section:	5.3.1.1
Page:	6 of 6
Change:	

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:



Date:	11/8/63
Section:	5.3.1.2
Page:	1 of 1
Change:	

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 $nHx_1x_2x_3\dots x_n$ 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 `10HbAVERAGEb=,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

`bAVERAGEb=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.

Date:	11/8/63
Section:	5.3.1.3
Page:	1 of 1
Change:	

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.

Date:	7/10/64
Section:	5.3.1.4
Page:	1 of 2
Change:	1

Thus for F and I fields the relation

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

holds for input and output.

Example:

Using the format `-lP,E+l3.7,F7.4`, ... the internal numbers 10.13 and 17.25 would print as

`+0101300E+03b17.250...`

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 . 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.

Date:	7/10/64
Section:	5.3.1.4
Page:	2 of 2
Change:	

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.

Date:	11/8/63
Section:	5.3.1.5
Page:	1 of 1
Change:	

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 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:

	0	1112	13	25	26	38	39	51
List	1	0 0	A	N	G			
List+1	1	0 0	B	M				
List+2	00	0 0	C	P				

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.

Date:	7/10/64
Section:	5.3.2
Page:	1 of 4
Change:	1

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
CALL   PRINT           In Program
...
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*
```

Date:	7/10/64
Section:	5.3.2
Page:	2 of 4
Change:	1

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,F1
                DECQ   1,B,1,0
F1            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:

Date:	7/10/64
Section:	5.3.2
Page:	3 of 4
Change:	1

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

CHAPTER 7. COMPILERS

TABLE OF CONTENTS

	Change	Date
7.1 FORTRAN II Version I		6/3/65
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(a ₁ , ..., a _n)		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

Date:	6/3/65
Section:	Chapter 7 Contents
Page:	1 of 1
Change:	ILLIAC II MANUAL

7. COMPILERS

7.1 FØRTRAN II, Version I

Version I of the FØRTRAN II compiler for ILLIAC II (hereinafter called FØRTRAN) 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 DØ loops because at least 15 bits are generally needed, and multiplication for array indexing is not moved outside of DØ 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 FØRTRAN compiler is a "one pass to assembly language" compiler. The output of the FØRTRAN 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).

FØRTRAN II for ILLIAC II is essentially compatible with FØRTRAN II for the IBM 7094 as implemented under the PØRTHØS operating system at the University of Illinois. There are certain differences, however. In particular, FØRTRAN 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 FØRTRAN; 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 FØRTRAN II. Cards in these categories will be ignored in compilation, but will appear on the listing, and will generate nonfatal errors.

Date:	6/3/65
Section:	7.1
Page:	1 of 3
Change:	
ILLIAC II MANUAL	

This brings us to another item in the philosophy of ILLIAC II as applied to FØRTRAN: 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 FØRTRAN 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 FØRTHØS.
2. If execution is desired after compilation, a \$ GØ card must be included between the ID card and the complete program.
3. Each program or subprogram written in FØRTRAN must be preceded by a \$ FØRTRAN 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 FØRTRAN. 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 FØRTRAN in general and want only to get at the peculiarities of this version of FØRTRAN. 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.

Date:	6/3/65
Section:	7.1
Page:	2 of 3
Change:	
ILLIAC II MANUAL	

Sections 7.1.1 through 7.1.6 specify in part the terms which can appear in FØRTRAN 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 FØRTRAN compilation.

Date:	6/3/65
Section:	7.1
Page:	3 of 3
Change:	
ILLIAC II MANUAL	

7.1.1 Characters Used

The (decimal) digits are 0 through 9; the letters are A through Z. The alphanumeric 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.

Date:	6/3/65
Section:	7.1.1
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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.

Date:	6/3/65
Section:	7.1.2
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

7.1.3 Constants

1. Fixed Point Constants: have 1 to 13 decimal digits, absolute value less than 2^{44} .
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 X 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.

Date:	6/3/65
Section:	7.1.3
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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 points constants.

Date:	6/3/65
Section:	7.1.4
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

7.1.5 Subscripted Variables

Fixed and floating point variables may both have as many as three subscripts attached e.g., AAAAAA(3), A11111(I, K, 4) are floating point subscripted variables; I11111(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).

$$C, V, V + C, V - C, C * V, C * V + C', C * V - C'$$

In particular, expressions such as $C + V$, $C * C'$ 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 $HOK(K, NOT(NOT(K)))$ 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.

Date:	6/3/65
Section:	7.1.5
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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 3HYES are not yet implemented except, of course, in FORMAT statements for output.

Date:	6/3/65
Section:	7.1.6
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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, \dots, 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,I), B(2,I) B(3,I), 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.

Date:	6/3/65
Section:	7.1.7
Page:	1 of 11
Change:	

ILLIAC II MANUAL

Example: $X = X + 1$. This will result in the contents of X being replaced by what is now in X, plus 1.

ASSIGN n TØ 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 GØ TØ statement, and n should appear in an assigned GØ TØ statement. Note that the compiler does not check to see that this last condition is satisfied.

Example: ASSIGN 47 TØ INTØ will cause INTØ in GØ TØ INTØ, (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 $a(a_1, \dots, a_n)$: see 7.1.8, 7.1.8.5.

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

CØMMØN 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 CØMMØN.

As each program is compiled, a counter is used to keep track of how much of CØMMØN 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 CØMMØN in the course of the compilation. Thus each variable or position of an array specified by a CØMMØN statement is associated with a unique number by this counter. When the programs are consolidated at run time into one program, every CØMMØN 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 CØMMØN are used by several subprograms.

Date:	6/3/65
Section:	7.1.7
Page:	2 of 11
Change:	
ILLIAC II MANUAL	

Example: If the first `COMMON` statements in each of two subprograms are

```
COMMON X1, X2
```

```
COMMON X3
```

and

```
COMMON R1, R2, R3
```

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), $i = 1, 2, 3, \dots$

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

```
COMMON A, B, C, D
```

```
EQUIVALENCE (C, G), (E, B)
```

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 `IOLIST`).

This can be accomplished by

```
DIMENSION XXX(256)
```

```
COMMON XXX
```

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.

Date:	6/3/65
Section:	7.1.7
Page:	3 of 11
Change:	
ILLIAC II MANUAL	

DIMENSION a_1, \dots, 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., $a_i = \text{MATRIX}(10, 20, 15)$ 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 = j_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) DIMENSION A(20)
 I = 2
 J = 20
 DO 10 K = 1, J, I
 10 A(K) = K ** 2

will result in A(K) being set equal to K^2 , K = 1, 3, 5, ..., 17, 19.

(b) DO 13 KAPPA = 3, 2
 13 A = KAPPA

will result in the floating point constant 3 being stored in A.

The DO statement may also be specified in the form DO n i = j₁, j₂ in which case it will be assumed that j₃ = 1.

i, j₁, j₂, j₃ should not be altered by any statements in the DO loop. However, altering i, j₂ or j₃ will be regarded as a nonfatal error. Altering j₁ has no effect. Care must be taken to see that j₃ is not set to zero, and that j₂ is not increased by more than j₃ each time through the loop. It must be true that j₁ ≥ 0, j₂ ≥ 0, and j₃ > 0.

Note that the last statement of a DO loop cannot be a nonexecutable statement (e.g., DIMENSION, COMMON), nor can it be any transfer (e.g., GO TO) or any DO statement. If the last statement in the DO loop would be one of these, the statement

n CONTINUE

where n is the statement number appearing in the DO statement, should be written following what would otherwise be the last statement. Aside from these restrictions, any FORTRAN statement may appear in a DO loop.

Example:

```

DIMENSION A(10), B(5)
DO 5 NICK = 1,5
A(2*NICK) = NICK
B(NICK) = NICK ** 3
PRINT 2, A(2* NICK), B(NICK)
2 FORMAT (1H0, 2(I10))
5 CONTINUE
END

```

Date:	6/3/65
Section:	7.1.7
Page:	5 of 11
Change:	

ILLIAC II MANUAL

This program will cause the following numbers to print out in the format shown (b represents a blank):

```
bbbbbbbbbb1bbbbbbbbbb1  
bbbbbbbbbb2bbbbbbbbbb8  
bbbbbbbbbb3bbbbbbbbbb27  
bbbbbbbbbb4bbbbbbbbbb64  
bbbbbbbbbb5bbbbbbbbbb125
```

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

Examples: (a) $A = 1$
 $D\emptyset$ 1 $I = 1, 5$
 $D\emptyset$ 1 $J = 1, 5$
 $A = I * J * A$
 1 $B = I + J + B$

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

(b) $D\emptyset$ 1 $I = 1, 10$
 $D\emptyset$ 2 $J = 1, 10$
 1 $I = I$
 2 $J = J$

is illegal, since the two $D\emptyset$ loops overlap.

Control can be transferred by means of any $G\emptyset$ $T\emptyset$ or IF statement from inside a $D\emptyset$ 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 $D\emptyset$ 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 $D\emptyset$ loop, unless the program changes it, in which case it will have whatever value the program gives it.

Date:	6/3/65
Section:	7.1.7
Page:	6 of 11
Change:	
ILLIAC II MANUAL	

However, execution of a subprogram inside a DO loop when the subprogram contains a DO loop with the same index as the original DO will not cause the index of the original DO loop to be changed. The general rule is that all variables in one program with same name (including indices of DO 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 FORTRAN 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

$\text{GO TO } n$

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 $(x_1, \dots, x_r), (y_1, \dots, y_s), \dots, (z_1, \dots, z_t)$: causes the variables x_1, \dots, x_r to be stored in the same location, y_1, \dots, y_s to be stored in the same location, \dots, z_1, \dots, z_t to be stored in the same location. Each x_i, y_i or z_i 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 $x_1 = A(3)$. Then if A is an array name, x_1 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, x_1 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

```
DIMENSION A(3,2), X(2,4)
EQUIVALENCE (A(5), X(4))
```

Date:	6/3/65
Section:	7.1.7
Page:	7 of 11
Change:	
ILLIAC II MANUAL	

appear in a program, then the arrays A and X will be stored overlapping each other as given by the figure:

A(1,1)	A(2,1)	A(3,1)	A(1,2)	<table border="1"><tr><td>A(2,2)</td></tr></table>	A(2,2)	A(3,2)				
A(2,2)										
X(1,1)	X(2,1)	X(1,2)	<table border="1"><tr><td>X(2,2)</td></tr></table>	X(2,2)	X(1,3)	X(2,3)	X(1,4)	X(2,4)		
X(2,2)										

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.

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

FREQUENCY: ignored.

FUNCTION $a(a_1, \dots, a_n)$: see 7.1.8.2.2

GOTO n: results in a transfer to the statement numbered n.

GOTO $(n_1, \dots, n_k), i$: This is a computed GOTO. 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, GOTO (3, 39, 14), I results in a transfer to statement 39.

GOTO $i, (n_1, \dots, n_k)$: This is an assigned GOTO. It is generally not as useful as a computed GOTO. 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.

Date: 6/3/65
Section: 7.1.7
Page: 8 of 11
Change:
ILLIAC II MANUAL

Example: If the statement ASSIGN 47 TO INTØ was the last ASSIGN statement referring to INTØ to be executed before GØ TØ INTØ, (16, 47, 36, 9), then the latter statement will result in a transfer to statement 47.

IF ACCUMULATOR ØVERFLOW n_1, n_2 : results in a transfer to statement n_1 if ØV (the accumulator overflow switch) is set, and to statement n_2 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 FPIIA. The user is referred to the ILLIAC II library write-ups for details.

IF (e) n_1, n_2, n_3 : results in a transfer to statement n_1, n_2 or n_3 depending whether the arithmetic expression e has a value less than, equal to, or greater than zero, respectively.

IF (SENSE LIGHT k) n_1, n_2 : If sense light k ($1 \leq k \leq 13$) is on, it will be turned off, and control will transfer to statement n_1 . If sense light k is off, it will remain off, and control will transfer to statement n_2 . If k is > 4 , then a nonfatal error message will be produced for the sake of compatability with FØRTRAN 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) n_1, n_2 : If sense switch k ($1 \leq k \leq 13$) has been turned on control will transfer to statement n_1 , and sense switch k will remain on. If sense switch k is turned off control will transfer to statement n_2 , 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 nonfatal error will be generated for the sake of compatability with FØRTRAN II for the IBM 7094.

Date:	6/3/65
Section:	7.1.7
Page:	9 of 11
Change:	
ILLIAC II MANUAL	

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_F , list: causes BCD line images for printing to be written on the output tape. n_F must be the number of a F~~O~~R~~M~~A~~T~~ statement which specifies the format of every line image. "List" is explained at the start of this section.

PUNCH n_F , list: causes BCD card images for punching to be written on the output tape. n_F must be the number of a F~~O~~R~~M~~A~~T~~ statement which specifies the format of every card image. "List" is explained at the start of this section.

READ n_F , list: causes BCD card images to be read from the input tape. n_F must be the number of a F~~O~~R~~M~~A~~T~~ 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 C~~O~~M~~M~~O~~N~~ is used in the same program as READ TAPE, 256 words at the start of C~~O~~M~~M~~O~~N~~ must be set aside as a buffer area (see the explanation of the C~~O~~M~~M~~O~~N~~ statement).

Date: 6/3/65
Section: 7.1.7
Page: 10 of 11
Change:
ILLIAC II MANUAL

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 $1 \leq k \leq 13$, then only sense light k is turned on. If $5 \leq k \leq 13$, 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 $a(a_1, \dots, a_n)$: 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.

Date:	6/3/65
Section:	7.1.7
Page:	11 of 11
Change:	
ILLIAC II MANUAL	

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 \$ GØ card and preceding each subprogram or calling program with a \$ FØRTRAN 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 unsubscripted 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'etre 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 FØRTRAN, 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. The

Date:	6/3/65
Section:	7.1.8
Page:	1 of 4
Change:	
ILLIAC II MANUAL	

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 placeholder 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

Date:	6/3/65
Section:	7.1.8
Page:	2 of 4
Change:	
ILLIAC II MANUAL	

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. M13 is used to hold the sense lights.

Date:	6/3/65
Section:	7.1.8
Page:	3 of 4
Change:	
ILLIAC II MANUAL	

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).

Date:	6/3/65
Section:	7.1.8
Page:	4 of 4
Change:	
ILLIAC II MANUAL	

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 FØRTRAN 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 FØRTRAN 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 FØRTRAN, 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 XPERTF, INF, and INTGER are fixed point valued; XPERT, IRTF, PAD, and XAF are floating point valued.

Date:	6/3/65
Section:	7.1.8.1
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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 FØRTRAN).
2. Arithmetic statement functions
3. Function subprograms

FØRTRAN provides the following library functions for the user:

SQRTF(α) = $\sqrt{\alpha}$
ELØGF(α) = $\log_e \alpha$
BLØGF(α) = $\log_2 \alpha$
TLØGF(α) = $\log_{10} \alpha$
EXPF(α) = e^α
SINF(α) = $\sin(\alpha)$ (α in radians)
CØSF(α) = $\cos(\alpha)$ (α in radians)
TANHF(α) = $\tanh(\alpha)$ (α in radians)
ATANF(α) = $\arctan(\alpha)$ radians

Note: The final F is optional in all of the above names; thus SIN(α) means the same as SINF(α).

In addition, the following functions are "built-in" to FØRTRAN (α and β should be floating point unless otherwise specified):

ABSF(α) = $|\alpha|$
XABSF^①
MØDF(α, β) = $\alpha - [\alpha/\beta]^\ominus \beta$
XMØDF
INTF(α) = $[\alpha]^\ominus$
XINTF
SIGNF(α, β) = $|\alpha|$ if $\beta \geq 0$
= $-|\alpha|$ if $\beta < 0$
XSIGNF

Date:	6/3/65
Section:	7.1.8.2
Page:	1 of 2
Change:	
ILLIAC II MANUAL	

FLØTF(α)	= α (change from fixed to floating)	} Not necessary on ILLIAC II
XFIXF(α)	= α (change from floating to fixed)	
DIMF(α, β)	= $\alpha - \text{MIN}(\alpha, \beta)$	
XDIMF		
MAXOF(α, β) ^③	= maximum of α and β where α and β are fixed point	
XMAXOF ^③		
MAXLF(α, β) ^③	= maximum of α and β where α and β are floating point	
XMAXLF ^③		
MINOF ^③	} defined by analogy with MAX.	
XMINOF ^③		
MINLF ^③		
XMINLF ^③		

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

(2) $[\alpha]$ = 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 MAXØF) 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.

Date:	6/3/65
Section:	7.1.8.2
Page:	2 of 2
Change:	
ILLIAC II MANUAL	

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

$$F(X,Y) = (X * Y - X ** Y)/(X + Y + 3.0)$$

defines a function F of two variables; the statement

$$G(U,V,W) = F(\text{SIN}(U ** 2 + V ** 2), W) + F(\text{COS}(U ** 2 + V ** 2), W)$$

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

$$F_1(X) = X(10)$$

$$F_2(X) = X(I)$$

$$F_3(X,J) = X(J)$$

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:

$$\begin{aligned} 1. \quad & F(0) = 1 \\ & F(X) = F(X - 1) * X \end{aligned}$$

$$\begin{aligned} 2. \quad & G(0) = 0 \\ & F(0) = 1 \\ & F(X) = G(X - 1) + X \\ & G(X) = F(X - 1) * X \end{aligned}$$

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.

Date:	6/3/65
Section:	7.1.8.2.1
Page:	2 of 2
Change:	
ILLIAC II MANUAL	

7.1.8.2.2 FUNCTION $a(a_1, \dots, a_n)$

Here a is a fixed or floating point name and (a_1, \dots, 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~~ $a(a_1, \dots, a_n)$ 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 ~~F~~ORTRAN program.

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

1. The ~~F~~ORTRAN 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 ~~D~~'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.

Date:	6/3/65
Section:	7.1.8.2.2
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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, IOLIST 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.

Date:	6/3/65
Section:	7.1.8.2.3
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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.

Date:	6/3/65
Section:	7.1.8.3
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

7.1.8.4 How to Define a Subroutine

FØRTRAN does not provide any built-in subroutines. Subroutines are defined by writing a subprogram which begins with SUBRØUTINE $a(a_1 \dots a_n)$, (a is the name of the subroutine; $(a_1 \dots a_n)$ is the list of dummy parameters: it may be empty) and which consists of any legal sequence of FØRTRAN 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.

Date:	6/3/65
Section:	7.1.8.4
Page:	1 of 1
Change:	
ILLIAC II MANUAL	

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 $a(a_1 \dots a_n)$. Then to call the subprogram a with call parameters $(b_1 \dots b_n)$, one writes the statement $CALL a(b_1 \dots b_n)$ in the calling program. This will result in automatic transfer to, execution of, and return from the subprogram using the parameters $b_1 \dots b_n$ in place of $a_1 \dots a_n$. 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.

Example: The following function subprogram computes $Z = \sum_{I=1}^N X(I)$ for values of $N \leq 20$:

Date:	6/3/65
Section:	7.1.8.5
Page:	1 of 3
Change:	
ILLIAC II MANUAL	

```

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

$$\sum_{i=1}^k i, \quad k = 1, 2, \dots, 15:$$

```

DIMENSION A (20)
B          = 2.0
DO 10 K    = 1,15
A(K)      = K
X         = PWRPLS(A,B,K)
10 PRINT 12,X
12 FORMAT (1H0, F30)
END

```

Date: 6/3/65 Section: 7.1.8.5 Page: 2 of 3 Change: ILLIAC II MANUAL

The output from this program is as follows:

```
2
8
64
1024
32768
2097152
268435456
68719476736
35184372088484
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.

Date:	6/3/65
Section:	7.1.8.5
Page:	3 of 3
Change:	
ILLIAC II MANUAL	

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 \$GO 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 FØRTRAN 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 FØRTRAN 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.

Date:	6/3/65
Section:	7.1.9
Page:	2 of 4
Change:	
ILLIAC II MANUAL	

- (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 $D\emptyset$ statements (e.g., 13 appears in $D\emptyset$ 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 $D\emptyset$ statement that it appears in in the user's program; the octal address given for each $D\emptyset$ statement appearance will be the location of the start of the $D\emptyset$ loop to which the $D\emptyset$ statement refers. Further, if a reference is made

Date:	6/3/65
Section:	7.1.9
Page:	3 of 4
Change:	
ILLIAC II MANUAL	

in a transfer statement (e.g., in an IF statement or in a GØ TØ statement) to a statement number appearing in a DØ statement, then that statement number will appear here once more than the number of DØ 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 DØ loop (this address will be greater than any of the addresses associated with appearances of the statement number in a DØ statement).

- (h) 1. ~~CØMMØN~~: location given is the start of the ~~CØMMØN~~ 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 16,000₈ through 17,777₈ (the four high blocks of core).
4. ~~PRØGRAM 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. ~~LOCATIONS ØF 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.

Date:	6/3/65
Section:	7.1.9
Page:	4 of 4
Change:	
ILLIAC II MANUAL	

CHAPTER 8. THE PROGRAM LIBRARY

TABLE OF CONTENTS

	Change	Date
8.1 Introduction		3/05/63
8.2 Classification		3/05/63
8.3 Subroutine Conventions	1	7/10/64
8.4 Program Descriptions		3/05/63
8.4-B1-ATAN1		7/20/64
8.4-B1-SIN1		7/20/64
8.4-B3-COSHL		7/29/64
8.4-B3-EXPL		7/29/64
8.4-B3-LGT1		11/26/63
8.4-B3-SINHL		7/29/64
8.4-B4-SQR1		10/10/63
8.4-D1-GQUL		11/19/63
8.4-D2-RKGL		7/15/64
8.4-E1-DVDF1		7/20/64
8.4-E1-LAG6		11/15/63
8.4-E1-LGUN		11/15/63
8.4-F4-SLQ1		7/20/64
8.4-G5-RAN1		8/10/64
8.4-J6-TOPS	2	11/14/64
8.4-KO-IOLIST		7/20/64
8.4-MO-CMP1		9/22/64
8.4-M2-PRINT		7/16/64

Date:	6/24/65
Section:	Chapter 8 Contents
Page:	1 of 1
Change:	5
ILLIAC II MANUAL	

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.

Date:	3/5/63
Section:	8.1
Page:	1 of 1
Change:	

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.

Date:	3/5/63
Section:	8.2
Page:	1 of 1
Change:	

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 JLH3. 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.

Date:	7/10/64
Section:	8.3
Page:	1 of 2
Change:	1

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 NICAP 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.

Date:	7/10/64
Section:	8.3
Page:	2 of 2
Change:	1

DIGITAL COMPUTER LABORATORY
 UNIVERSITY OF ILLINOIS
 URBANA, ILLINOIS
 ILLIAC II LIBRARY PROGRAM
 B1-ATAN1-OO-UI-AL

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 μ sec (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:

No.	Interval	Comments
0	$0 \leq X < \tan \frac{\pi}{24}$	Use P(x) directly
1	$\tan \frac{\pi}{24} \leq X < \tan \frac{3\pi}{24}$	Use Eqn A with K = 1
2	$\tan \frac{3\pi}{24} \leq X < \tan \frac{5\pi}{24}$	Use Eqn A with K = 2
3	$\tan \frac{5\pi}{24} \leq X < \tan \frac{7\pi}{24}$	Use Eqn A with K = 3
4	$\tan \frac{7\pi}{24} \leq X < \tan \frac{9\pi}{24}$	Use Eqn A with K = 4
5	$\tan \frac{9\pi}{24} \leq X < \tan \frac{11\pi}{24}$	Use Eqn A with K = 5
6	$\tan \frac{11\pi}{24} \leq X < \infty$	Use Eqn B

Programmed by: John Kelly
 Approved by:
John Kelly

Date: 7/20/64
 Section: 8.4-B1-ATAN1
 Page: 1 of 5
 Change:

METHOD (Continued): Eqn A: $\arctan X = \frac{k\pi}{12} + \arctan t_k, \quad t_k = \frac{X - \tan \frac{k\pi}{12}}{1 + X \tan \frac{k\pi}{12}}$

Eqn B: $\arctan X = \frac{\pi}{2} - \arctan \frac{1}{X}$

P(x) is used to compute $\arctan t_k$ and $\arctan \frac{1}{X}$.

$$P(x) = a_1x + a_3x^3 + a_5x^5 + \dots + a_{17}x^{17}$$

where

$$a_1 = 1.00000 \ 00000 \ 00000$$

$$a_3 = -0.33333 \ 33333 \ 33333$$

$$a_5 = 0.19999 \ 99999 \ 99998$$

$$a_7 = -0.14285 \ 71428 \ 56331$$

$$a_9 = 0.11111 \ 11109 \ 07794$$

$$a_{11} = -0.09090 \ 90609 \ 63368$$

$$a_{13} = 0.07692 \ 04073 \ 24915$$

$$a_{15} = -0.06652 \ 48229 \ 41311$$

$$a_{17} = 0.05467 \ 21009 \ 39594$$

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: Perlin, I. E. and J. R. Garrett. Mathematics of Computation, National Academy of Sciences--National Research Council, vol. 14, No. 71, July 1960, pp. 270-274.

Date:	7/20/64
Section:	8.4-B1-ATAN1
Page:	2 of 5
Change:	

ATANI	SFR	5,COMMON	SAVE F5.	ATANI 01
	SFR	2,COMMON+2	SAVE F2.	ATANI 02
	SFR	3,COMMON+3	SAVE F3.	ATANI 03
	TN	ATANIG	WANT ABSOLUTE VALUE OF X.	ATANI 04
	STR	F2	SAVE X.	ATANI 05
	CAM	4	SET POSITIVE SIGN FLAG.	ATANI 06
ATANIA	CNM	5	SET M5 FOR INTERVAL FLAG.	ATANI 07
ATANIB	SUB	ATANIK+M5+1	FORM NEXT DIFFERENCE TO GET INTERVAL.	ATANI 08
	ADM	5,1	INCREMENT INTERVAL FLAG.	ATANI 09
	TZP	ATANIB	JUMP IF INTERVAL NOT YET REACHED.	ATANI 10
	JZM	5,ATANIC	JUMP IF INTERVAL 0 TO EXECUTION OF POLY.	ATANI 11
	CAM	6,M5-6	TEST FOR INTERVAL 6.	ATANI 12
	JZM	6,ATANIH	JUMP IF INTERVAL 6.	ATANI 13
	CAD	F2	ENTER X.	ATANI 14
	SUB	ATANIL+M5-1	FORM $X - \tan(K \cdot \pi/12)$.	ATANI 15
	STR	F3	STORE NUMERATOR $X - \tan(K \cdot \pi/12)$.	ATANI 16
	CAD	ATANIL+M5-1	ENTER $\tan(K \cdot \pi/12)$.	ATANI 17
	MPY	F2	FORM THE DENOMINATOR	ATANI 18
	ADD	1.	$1 + X * \tan(K \cdot \pi/12)$.	ATANI 19
	VID	F3	$T(K) = (X - \tan(K \cdot \pi/12)) / (1 + X * \tan(K \cdot \pi/12))$	ATANI 20
	SFR	6,COMMON+1	SAVE F6.	ATANI 21
	LFR	6,ATANIM+M5-1	STORE $K \cdot \pi/12$ IN F6.	ATANI 22
	STR	F2	SAVE $T(K)$ IN F2.	ATANI 23
ATANIC	CAD	F2	ENTER X, $T(K)$, OR $1/X$. CALL IT Y.	ATANI 24
	MPY	F1	FORM $Y**2$.	ATANI 25
	CAM	6,ATANIJ	SET LOCATION OF FIRST COEFFICIENT.	ATANI 26
	CSM	7,8	SET POLY ENDTEST COUNTER.	ATANI 27
	STR	F3	SAVE $Y**2$ IN F3.	ATANI 28
	CAD	6,1,	ENTER FIRST COEFFICIENT $A(17)$.	ATANI 29
	FLD		POLY MEANS THE POLYNOMIAL EXPRESSION.	ATANI 30
	MPY	F3	FORM $POLY * Y**2$.	ATANI 31
	ADD	6,1,	FORM $POLY +$ NEXT COEFFICIENT.	ATANI 32

Date: 7/20/64
 Section: 8.4-BL-ATANI1
 Page: 3 of 5
 Change:

	CJF	7,	POLY ENDTST.	ATANI	33	
	MPY	F2	POLY=A(1)*Y+A(3)*Y**3+...+A(17)*Y**17.	ATANI	34	
	JZM	5, ATANI D	JUMP IF INTERVAL 0.	ATANI	35	
	JNM	5, ATANI I	JUMP IF INTERVAL 6.	ATANI	36	
	ADD	F6	ARCTAN(X) = POLY + K*PI/12.	ATANI	37	
	ATANI D	JZM	4, ATANI E	JUMP IF POSITIVE SIGN FLAG WAS SET.	ATANI	38
		STN	F0	X WAS NEGATIVE, SO MAKE	ATANI	39
		CAD	F0	ARCTAN(X) NEGATIVE.	ATANI	40
	ATANI E	JZM	5, ATANI F	JUMP IF INTERVAL 0.	ATANI	41
		LFR	6, COMMON+1	RESTORE F6.	ATANI	42
	ATANI F	LFR	5, COMMON	RESTORE F5.	ATANI	43
		LFR	2, COMMON+2	RESTORE F2.	ATANI	44
		LFR	3, COMMON+3	RESTORE F3.	ATANI	45
		JLH	M3	EXIT ATANI SUBROUTINE.	ATANI	46
	ATANI G	STN	F2	SAVE ABSOLUTE X.	ATANI	47
		CNM	4	SET NEGATIVE SIGN FLAG.	ATANI	48
		CAD	F2	ENTER X = ABSOLUTE X.	ATANI	49
		TRA	ATANI A	JUMP TO INTERVAL TEST.	ATANI	50
	ATANI H	CAD	1.	SET NUMERATOR.	ATANI	51
		DIV	F2	FORM Y = 1/X.	ATANI	52
		SFR	6, COMMON+1	SAVE F6.	ATANI	53
		CNM	5	SET INTERVAL MODIFIER M5 NEGATIVE.	ATANI	54
		LFR	6, ATANI N	STORE PI/2 IN F6.	ATANI	55
		STR	F2	SAVE 1/X IN F2.	ATANI	56
		TRA	ATANI C	JUMP TO EXECUTE POLY.	ATANI	57
	ATANI I	SUB	F6	-ARCTAN(X) = ARCTAN(1/X) - PI/2.	ATANI	58
		ADM	4, 1	REVERSE SIGN SINCE -ARCTAN(X) L ZERO.	ATANI	59
		TRA	ATANI D	JUMP TO EXIT.	ATANI	60
		FIL		THE FOLLOWING ARE THE TABLES USED.	ATANI	61
	ATANI J	OCTQ	6776, 17664, 13050, 1776, 15676, 725, 6430, 4777	A(17), A(15).	ATANI	62
		OCTQ	2354, 4162, 4733, 377, 15056, 10570, 6100, 11377	A(13), A(11).	ATANI	63
		OCTQ	3434, 7070, 15622, 3777, 13333, 6666, 15556, 11777	A(9), A(7).	ATANI	64
		OCTQ	6314, 14631, 11463, 3377, 15252, 12525, 5252, 12600	A(5), A(3).	ATANI	65
		OCTQ	2000, , , 1	A(1).	ATANI	66
	ATANI K	OCTQ	4154, 17723, 1623, 1577, 2205, 5730, 430, 2400	TABLE OF	ATANI	67

Date:	7/20/64
Section:	8.4-B1-ATANI
Page:	4 of 5
Change:	

	OCTQ	2646,5510,6660,5600,4223,505,16475,11600 DIFFERENCES	ATANI 68
	OCTQ	2161,12334,6504,10001,2456,7456,2032,12202 FOR	ATANI 69
	OCTQ	7777,17777,17777,17677 THE INTERVAL TEST.	ATANI 70
ATAN1L	OCTQ	2111,10242,17236,14600,4474,15164,5440,6400 TABLE	ATANI 71
	OCTQ	2000,1,3355,11727,4130,4601 OF	ATANI 72
	OCTQ	7355,11727,4130,4601 TAN(K=PI/12).	ATANI 73
ATAN1M	OCTQ	2060,5221,14055,7200,4140,12443,10132,16400 TABLE	ATANI 74
	OCTQ	6220,17665,4210,5600,2060,5221,14055,7201 OF	ATANI 75
	OCTQ	2474,6466,3070,15001 K=PI/12.	ATANI 76
ATAN1N	OCTQ	3110,7732,12104,2601 PI/2.	ATANI 77

Date:	7/20/64
Section:	8.4-B1-ATANI1
Page:	5 of 5
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

ILLIAC II LIBRARY PROGRAM

BL-SIN1-OO-UI-AL
BL-COS1-OO-UI-AL

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 μ 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 $\cos \pi(x - 1/2) = \sin \pi x$. $x(\text{mod } 2)$ 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

$$\sin x = \sum_{k=0}^6 C_k x^{2k+1}$$

Programmed by: Robert Lange
Revised by: Roberta White
Approved by: *ew hear*

Date: 7/20/64
Section: 8.4-BL-SIN1
Page: 1 of 3
Change:

METHOD (Continued): is evaluated by the standard technique. The coefficients were calculated on ILLIAC II, starting with

$$\sin y \sim y - \frac{y^3}{3!} + \dots - \frac{y^{19}}{19!}$$

REFERENCE: Hildebrand, F. B. Introduction to Numerical Analysis, McGraw Hill, New York (1956).

Date:	7/20/64
Section:	8.4-BL-SIN1
Page:	2 of 3
Change:	

	FLD				SIN10000
	DECQ	-6,COS1+5	MODIFIER CONSTANTS		SIN10001
SIN1	SUB	10,3,2048	COMPUTE SINE		SIN10002
COS1	STF	0,3,			SIN100 3
	SFR	6,COMMON	SAVE F6		SIN10004
	TOR	1,COS1+1	CLEAR OV		SIN10005
	DAV	10,3,2048	1X1 - 1/2 = A		SIN10006
	STN	F2	X = 1/2 - 1X1		SIN10007
	MPY	F0			SIN10008
	LFR	6,COS1-1	SET M8 AND M9, I = -6		SIN10009
	STR	F0	STORE X**2 IN F0		SIN10010
	CAD	9,1,	C7 = A		SIN10011
	MPY	F0	*X**2		SIN10012
	ADD	9,1,	+ C-I		SIN10013
	CJF	8,1,	IS I =0		SIN10014
	LFR	6,COMMON	RESTORE F6		SIN10015
	MPY	F2	*X		SIN10016
	JLH	3,,	= EXIT		SIN10017
	OCTQ	3517,6174,4521,15173	C7		SIN10018
	OCTQ	14165,15472,10711,10575	C6		SIN10019
	OCTQ	2501,15532,14354,1577	C5		SIN10020
	OCTQ	13151,6467,17623,6400	C4		SIN10021
	OCTQ	5063,5706,6336,13601	C3		SIN10022
	OCTQ	15325,1030,14735,4602	C2		SIN10023
	OCTQ	6220,17665,4210,14201	C1		SIN10024
	COMMON BSS	1			

Date:	7/20/64
Section:	8.4-BI-SIN1
Page:	3 of 3
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
B3-COSHL-OO-UI-AL

NAME: Hyperbolic cosine

OTHER SUBROUTINES USED: EXPl

TEMPORARY STORAGE;< None

NUMBER OF WORDS: 5 words

EXECUTION TIME: 161.4 μ sec (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 7.9×10^{-12} and the average error is 1.0×10^{-12} .

Programmed by: John Kelly
Approved by:

J. W. Kelly

Date: 7/29/64
Section: 8.4-B3-COSHL
Page: 1 of 2
Change:

COSH1 SFR 4,COSH1A
CALL EXP1
STR F0
VID 1.
ADD F0
MPY 10,3,2048
LFR 4,COSH1A
JLH M3
COSH1A BSS 1

SAVE F4.
COSH1(X) IS
COMPUTED
FROM THE
EXP1 SUBROUTINE.
 $COSH1(X) = (1/2) * (E^{**X} + E^{**(-X)})$.
RESTORE F4.
EXIT COSH1 SUBROUTINE.
COSH1 TEMP STORAGE.

COSH1 01
COSH1 02
COSH1 03
COSH1 04
COSH1 05
COSH1 06
COSH1 07
COSH1 08
COSH1 09

Date: 7/29/64
Section: 8.4-B3-COSH1
Page: 2 of 2
Change:

DIGITAL COMPUTER LABORATORY
 UNIVERSITY OF ILLINOIS
 URBANA, ILLINOIS
 ILLIAC II LIBRARY PROGRAM
 B3-EXPl-OO-UI-AL

NAME: Exponential

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE: None

NUMBER OF WORDS: 40 words

EXECUTION TIME: 127.5 μ sec

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 $x \geq 127 \log 2$ [$\approx 88^*$.]

MATHEMATICAL METHOD: $e^x = K^w$ where $K = 2^{1/8}$, $w = x \log_K e$. $w = 16a + b + f$ 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

$$F(z) = 2 - z + \frac{1}{6} z^2 - \frac{1}{360} z^4 + \frac{1}{15120} z^6 + \dots$$

So

$$e^{cf} = 2\left(\frac{1}{2} + \frac{f}{\frac{2}{c} - f + \frac{c}{6} f^2 - \frac{c^3}{360} f^4 + \frac{c^5}{15120} f^6}\right)$$

*There is a correct 8-bit exponent in the accumulator. This may be too large to store.

Programmed by: N. T. Hamilton

Approved by:

L. W. Gear

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 3.9×10^{-13} .

Date:	7/29/64
Section:	8.4-B3-EXP1
Page:	2 of 4
Change:	

EXP1	TOR	2,EXP1	CLEAR OV.	EXP1	01
	MPY	EXP1F	FORM $W = X * \text{LOG2}E$. $E**X = K**W$.	EXP1	02
	SFR	4,EXP1E	SAVE F4	EXP1	03
	SFR	5,EXP1E+1	AND F5.	EXP1	04
	SAM	F5	$W = 16*A + B + F$.	EXP1	05
	LIN	8	SHIFT AND	EXP1	06
	SEQ	F0	STORE A IN M0	EXP1	07
	SAM	F4	AND B IN M1.	EXP1	08
	TNOR	EXP1C	JUMP IF W NOT TOO EXTREME.	EXP1	09
	JNM	4,EXP1B	JUMP TO SET UF IF X NEGATIVE.	EXP1	10
EXP1A	DIV	15,3,	SET OV.	EXP1	11
EXP1B	CAD	15,3,	CLEAR ACC OR SET UF.	EXP1	12
	LFR	4,EXP1E	RESTORE F4	EXP1	13
	LFR	5,EXP1E+1	AND F5.	EXP1	14
	JLH	M3	EXIT EXP1 SUBROUTINE.	EXP1	15
EXP1C	AND	EXP1G	MASK W TO LEAVE FRACTIONAL	EXP1	16
	STR	F5	PART F AND SAVE IN F5.	EXP1	17
	MPY	F0	FORM $F**2$.	EXP1	18
	CAM	2,EXP1H	SET LOCATION OF POLY COEFFICIENTS.	EXP1	19
	STR	F0	SAVE $F**2$.	EXP1	20
	MPY	2,1,	FORM THE POLY	EXP1	21
	ADD	2,1,	EXPANSION FOR	EXP1	22
	MPY	F0	$(1/2) * K**F$, $K = 2**(1/8)$.	EXP1	23
	CRM	1,9	RIGHT JUSTIFY B.	EXP1	24
	ADD	2,1,	FORM	EXP1	25
	MPY	F0	POLY.	EXP1	26
	ANM	1,15	MASK M1 TO LEAVE B.	EXP1	27
	ADD	M2	CONTINUE	EXP1	28
	SUB	F5	FORMING	EXP1	29
	VID	F5	POLY.	EXP1	30
	ANN	,8064	TEST FOR OV. LEFT 6 BITS	EXP1	31
	CAM	2	MUST BE 0 TO PASS.	EXP1	32
	ORM	,3,127	TEST FOR UF. LEFT 6 BITS	EXP1	33
	CNM	3	MUST BE 1 TO PASS.	EXP1	34
	JZM	2,EXP1D	JUMP IF NO OV.	EXP1	35

Date:	7/29/64
Section:	8.4-B3-EXP1
Page:	3 of 4
Change:	

	JZM	3,EXP1D	JUMP IF NO UF.	EXP1	36
	JPM	,EXP1A	X POSITIVE SO SET OV.	EXP1	37
	TRA	EXP1B	X NEGATIVE SO SET UF.	EXP1	38
EXP1D	ADD	10,3,2048	POLY EXPANSION COMPLETED.	EXP1	39
	CAE	MO	PUT A IN EXPONENT REGISTER.	EXP1	40
	MPY	M1+EXP1I	MPY BY 2 * K**B.	EXP1	41
	TRA	1,EXP1B	JUMP TO EXIT.	EXP1	42
EXP1E	BSS	2	EXP1 TEMP STORAGE.	EXP1	43
EXP1F	OCTQ	05612,12166,05127,00202	LOG2E.	EXP1	44
EXP1G	OCTQ	00000,00777,17777,17600	MASK.	EXP1	45
EXP1H	OCTQ	00000,00000,00026,03001	POLY	EXP1	46
	OCTQ	17777,17760,15377,06401	EXPANSION	EXP1	47
	OCTQ	00016,14460,03775,04201	COEFFICIENTS.	EXP1	48
	OCTQ	02705,05073,02453,10203		EXP1	49
EXP1I	OCTQ	04000,00000,00000,00001	TABLE OF	EXP1	50
	OCTQ	04271,05603,14372,12001	2 * 2**(B/8)	EXP1	51
	OCTQ	04603,07740,12143,07001	FOR 0 .LE. B .LE. 15.	EXP1	52
	OCTQ	05137,16655,05154,05001		EXP1	53
	OCTQ	05520,04746,06376,07401		EXP1	54
	OCTQ	06126,07124,02125,04001		EXP1	55
	OCTQ	06564,04771,11265,12401		EXP1	56
	OCTQ	07254,00615,14767,04401		EXP1	57
	OCTQ	02000,00000,00000,00002		EXP1	58
	OCTQ	02134,12701,16175,05002		EXP1	59
	OCTQ	02301,13760,05061,13402		EXP1	60
	OCTQ	02457,17326,12466,02402		EXP1	61
	OCTQ	02650,02363,03177,03602		EXP1	62
	OCTQ	03053,03452,01052,12002		EXP1	63
	OCTQ	03272,02374,14532,15202		EXP1	64
	OCTQ	03526,00306,16373,12202		EXP1	65

Date:	7/29/64
Section:	8.4-B3-EXPI
Page:	4 of 4
Change:	

UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
URBANA, ILLINOIS

ILLIAC II LIBRARY PROGRAM

B3-LGT1-00-UI-AL
B3-LGE1-00-UI-AL
B3-LGB1-00-UI-AL

NAME: Logarithm

TEMPORARY STORAGE: COMMON, F2

OTHER SUBROUTINES USED: None

NUMBER OF WORDS: 53

EXECUTION TIME: 170 microseconds

ENTRY: Standard by

CALL LGT1 for $\log_{10}x$

CALL LGE1 for $\log_e x$

CALL LGB1 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 $x \leq 0$, 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 $x = f \cdot 4^n$, $1/4 \leq f < 1$. An appropriate value of

$$\frac{16}{4^{\frac{1}{2} + k}} \quad (k = 0, 1, \dots, 11)$$

is chosen from a stored table of values so that

Programmed by: Marvin Gaer
Approved by:

M. T. Hamilton

Date: 11/26/63
Section: 8.4-B3-LGT1
Page: 1 of 5
Change:

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 $\log_e f^1$:

$$\log_e(f^1) = \log_e(1 + f^{11}) = \log_e\left(\frac{1+a}{1-a}\right) = 2a + \frac{2a^3}{3} + \frac{2a^5}{5} + \dots + \frac{2a^{13}}{13}$$

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.

$$\log_{10} x = (\log_e 10)^{-1} (\log_e x); \quad \log_2 x = (\log_e 2)^{-1} (\log_e x).$$

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.

Date:	11/26/63
Section:	8.4-B3-LGT1
Page:	2 of 5
Change:	

LGT1	FIL	4,COMMON	LOG 10 X	LGT10000
	SFR	1,1	M1= 1 MEANS LOG 10 X	LGT10001
	CAM	3,LGT1+3		LGT10002
LGB1	TRA	4,COMMON	LOG 2 X	LGT10003
	SFR	1	M0=0 MEANS LOG 2 X	LGT10004
	CAM	3,LGT1+3		LGT10005
LGE1	TRA	4,COMMON		LGT10006
	SFR	1,1	M1=-1 MEANS LOG E X	LGT10007
	CSM	3,LGT1+17	X LT EQUAL TO 0 MEANS ERROR	LGT10008
	TZN	F2	NORMALIZE X, X=F-4N	LGT10009
	STR	3		LGT10010
	SEX	0		LGT10011
	CAE	2	16-F FOR	LGT10012
	ADE	0	TABLE LOOKUP	LGT10013
	SIA	2	RESTORE F	LGT10014
	SBE	LGT1+15+M0	.16/4.5+K=F	LGT10015
	MPY	1.	F 11 = F 1 - 1	LGT10016
	SUB	F2	STORE F 11	LGT10017
	STR	2.		LGT10018
	ADD	F2	A= F 11 / (2+F 11)	LGT10019
	VID	F2	STORE A	LGT10020
	STR	F2	XQ=A2	LGT10021
	MPY	F3	STORE A2 IN F3	LGT10022
	STR	2,-6		LGT10023
	CAM	LGT1+43	A6	LGT10024
	CAD			LGT10025
	FIL			LGT10026
	MPY	F3	XA 2	LGT10027
	ADD	LGT1+50+M2	+A1, I=5,000	LGT10028
	CJF	2,0,	LOG	LGT10029
	MPY	F2	.A=LOG(1+F 11)=LOG F 1	LGT10030
	SUB	LGT1+27+M0	-LOG 16 / 4.5+N=LN F	LGT10031
	STR	F2	STORE LN F	LGT10032
	CAD	LGT1+50	LN 4	LGT10033
	MPY	M3.	N.LN4	LGT10034

Date: 11/26/63
 Section: 8.4-B3-LG11
 Page: 3 of 5
 Change:

ADD	F2	+LN F= LN F.4N=LNx	LGT10035
JZM	1,3,LGT1+16		LGT10036
JNM	1,LGT1+16	LOG E	LGT10037
MPY	LGT1+51	.1/ LOG E 10	LGT10038
LFR	4,COMMON	RESTORE F4	LGT10039
JLH	M3		LGT10040
MPY	LGT1+52	.1 / LN 2 = LOG 2 X	LGT10041
TRA	LGT1+16		LGT10042
DIV	15,3,		LGT10043
TRA	LGT1+16		LGT10044
FIL			LGT10045
OCTQ	07070,16161,14343,11401	SCALE= 16 / 4.5+R	LGT10046
OCTQ	05642,16427,04272,02601		LGT10047
OCTQ	04730,11661,03542,07001		LGT10048
OCTQ	04210,10421,01042,01601		LGT10049
OCTQ	03607,10360,17036,01201		LGT10050
OCTQ	03274,12065,16241,12601		LGT10051
OCTQ	03030,06060,14141,07601		LGT10052
OCTQ	02620,13102,14413,02001		LGT10053
OCTQ	02436,13412,03656,01201		LGT10054
OCTQ	02275,12045,16641,02601		LGT10055
OCTQ	02151,16713,00432,07601		LGT10056
OCTQ	02041,00410,04102,01001		LGT10057
OCTQ	02422,17224,04027,11201	LOG 16 / 4.5+R	LGT10058
OCTQ	02105,07400,07163,01601		LGT10059
OCTQ	07151,11744,16071,16000		LGT10060
OCTQ	06037,07541,05534,00000		LGT10061
OCTQ	05036,14777,02762,04400		LGT10062
OCTQ	04127,03556,02640,12600		LGT10063
OCTQ	03275,04512,11110,14600		LGT10064
OCTQ	02510,12560,03470,12200		LGT10065
OCTQ	07714,10706,14547,06177		LGT10066
OCTQ	05337,12006,13250,07777		LGT10067
OCTQ	03114,15362,16531,05577		LGT10068
OCTQ	04040,12730,11716,10576		LGT10069

Date:	11/26/63
Section:	8.4-B3-LGT1
Page:	4 of 5
Change:	

OCTQ	04730, 11661, 03542, 10377	A6	LGT10070
OCTQ	05642, 16427, 04272, 03377	A5	LGT10071
OCTQ	07070, 16161, 14343, 10577	A4	LGT10072
OCTQ	02222, 04444, 11111, 02200	A3	LGT10073
OCTQ	03146, 06314, 14631, 11400	A2	LGT10074
OCTQ	05252, 12525, 05252, 12600	A1	LGT10075
OCTQ	04000, 00000, 00000, 00001	A0	LGT10076
OCTQ	02613, 11027, 17372, 03601	LN 4	LGT10077
OCTQ	03362, 15730, 12446, 16400	1 / EN 10	LGT10078
OCTQ	02705, 05073, 02453, 10201	WILL BE 1 / EN 2	LGT10079

Date:	11/26/63
Section:	8.4-B3-LGPI1
Page:	5 of 5
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
B3-SINH1-00-UI-AL

NAME: Hyperbolic sine

OTHER SUBROUTINES USED: EXPl

TEMPORARY STORAGE: None

NUMBER OF WORDS: 21 words

EXECUTION TIME: 89.3 μ sec for $|x| \leq 1$, 178.3 μ 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 $\sinh^2(2x) = 4\sinh^2x(1 + \sinh^2x)$, the maximum relative error is 2.2×10^{-12} and the average error is $.30 \times 10^{-12}$.

REFERENCE: Clenshaw, C. W., Chebyshev Series for Mathematical Functions, Vol. 5 of Mathematical Tables, National Physical Laboratory, p. 24.

Programmed by: John Kelly
Approved by:



Date: 7/29/64
Section: 8.4-B3-SINH1
Page: 1 of 3
Change:

SINH1	SFR	4, SINH1C	SAVE F4.	SINH1 01
	STR	F4	SAVE X.	SINH1 02
	DAV	.	FORM	SINH1 03
	SUB	1.	ABS.(X) - 1.	SINH1 04
	TP	SINH1B	JUMP IF X .L. -1 OR X .G. 1.	SINH1 05
	CAD	F4	FORM	SINH1 06
	MPY	F1	X**2.	SINH1 07
	SFR	5, SINH1C+1	SAVE F5.	SINH1 08
	CAM	4, SINH1D	SET LOCATION OF POLYNOMIAL CONSTANTS.	SINH1 09
	STR	F0	Y = X**2.	SINH1 10
	MPY	4, 1,	FORM A13 * Y.	SINH1 11
	CSM	5, 5	SET LOOP COUNTER.	SINH1 12
	FLD			SINH1 13
	ADD	4, 1,	SINH1(X) IS	SINH1 14
	MPY	F0	COMPUTED FROM	SINH1 15
	CJF	5	A CHEBYSHEV	SINH1 16
	ADD	M4	POLYNOMIAL EXPANSION.	SINH1 17
	MPY	F4	SINH1(X) = A1*X+A3*X**3+...+A13*X**13.	SINH1 18
	LFR	5, SINH1C+1	RESTORE F5.	SINH1 19
SINH1A	LFR	4, SINH1C	RESTORE F4.	SINH1 20
	JLH	M3	EXIT SINH1 SUBROUTINE.	SINH1 21
SINH1B	CAD	F4	X.	SINH1 22
	CALL	EXP1	SINH1(X) IS	SINH1 23
	STR	F0	COMPUTED	SINH1 24
	VID	-1.	FROM THE	SINH1 25
	ADD	F0	EXP1 SUBROUTINE.	SINH1 26
	MPY	10, 3, 2048	SINH1(X) = (1/2) * (E**X - E**(-X)).	SINH1 27
	TRA	SINH1A	JUMP TO EXIT.	SINH1 28
SINH1C	BSS	2	SINH1 TEMP STORAGE.	SINH1 29
SINH1D	OCTQ	05474, 00102, 04236, 16560	A13.	SINH1 30
	OCTQ	03271, 04232, 12511, 15164	A11.	SINH1 31
	OCTQ	05616, 17127, 11460, 16767	A9.	SINH1 32

Date:	7/29/64
Section:	8.4-B3-SINH1
Page:	2 of 3
Change:	

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

Date: 7/29/64
Section: 8.4-B3-SIHNL
Page: 3 of 3
Change:

UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
ILLIAC II LIBRARY PROGRAM
B4-SQR1-00-UI-AL

NAME: Square Root

PURPOSE: Replaces the rounded contents of the accumulator with its square root if the accumulator is positive. Does nothing if negative. This routine always clears overflow.

OTHER SUBROUTINES USED: None

TEMPORARY STORAGE USED: F2 and COMMON

NUMBER OF WORDS: 9

DURATION: 150 microseconds

ACCURACY: Maximum error is not more than 1.5 in the least significant place (bit 44).

RANGE USE: Accumulator exponent can be in range -128 to +127 on entry.

ENTRY: Standard by CALL SQR1 with X in the accumulator.

EXIT: Standard with \sqrt{X} in the accumulator if $X \geq 0$ or with X in the accumulator if $X < 0$.

METHOD: Four iterations of the Newton formula $X_{N+1} = (X_N + X/X_N)/2$ are used. The first approximation X_0 is formed from the normalized form of $X = Y4^E$ as follows:
If E odd, $X_0 = (Y + 1)4^{(E-1)/2}$.
If E even, $X_0 = 1/2(Y + 1)4^{E/2}$.
The iteration is done with a zero exponent.

Programmed by: *E. W. Gear*
Approved by:

Date: 10/10/63
Section: 8.4-B4-SQR1
Page: 1 of 2
Change:

SQR1

FIL
TZN 1, SQR1+8
SFR 4, COMMON
STR 0, 3,
SEX 3
CAE 0
STR 2, 3,
ADD 2, 3,
ADD 2,
SBE 1
CSM 2, 4
STR 0, 3,
VID 2, 3,
ASC 0, 3,
CAT 0, 3,
SBE 1
CJU 2, SQR1+4
CRM 3, 1
JPM 3, 2, SQR1+7
STR 0, 3,
CAT 0, 3,
ADE 3,
LFR 4, COMMON
TOR 3, SQR1+8
JLH 3,,

SQUARE ROOT FIRST CARD

TEST FOR ZERO
SAVE F4
ROUND
STORE EXPONENT
CLEAR EXPONENT

DOUBLE
ADD I
DIVIDE BY FOUR
SET COUNT
STORE XN
X/XN

DOUBLE
DIVIDE BY FOUR
LOOP FOUR TIMES
HALVE EXPONENT
TEST FOR EVEN EXPONENT
DOUBLE

SET EXPONENT
RESTORE F4
RESET OVERFLOW
EXIT

SQR10001
SQR10002
SQR10003
SQR10004
SQR10005
SQR10006
SQR10007
SQR10008
SQR10009
SQR10010
SQR10011
SQR10012
SQR10013
SQR10014
SQR10015
SQR10016
SQR10017
SQR10018
SQR10019
SQR10020
SQR10021
SQR10022
SQR10023
SQR10024

Date: 10/10/63
Section: 8.4-B4-SQR1
Page: 2 of 2
Change:

UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

ILLIAC, II Library Routine

J5-CCP4SC-40-UI-AL

August 19, 1965

Entry Name

CCP4SC

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 CCP6LN 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, FO-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

CALL CCP4SC
DECQ X,S,N,K
DECQ T,,

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

Date: 8/19/65
Section: 8.4-CCP4SC
Page: 1 of 2
Change: 0
ILLIAC II MANUAL

UNIVERSITY OF ILLINOIS
GRADUATE COLLEGE
DEPARTMENT OF COMPUTER SCIENCE

ILLIAC II LIBRARY ROUTINE

J5-CCP6LN-41-UI-AL

August 17, 1965

Entry Name

CCP6LN

IDENTIFICATION

CalComp Line for FORTRAN and NICAP.

PURPOSE

CCP6LN plots and connects on the CalComp Plotter a specified subset of the set of points

$(\frac{X(I)-XMIN}{DX}, \frac{Y(I)-YMIN}{DY})$, $I=1, 2, \dots, N$,
given the arrays X and Y, whose elements are X(1), X(2), ..., X(N) and Y(1), Y(2), ..., Y(N), respectively, and given XMIN, YMIN, DX and DY.

RESTRICTIONS

$(y_{max}-y_{min})/DY < 29$ where y_{max} is the maximum and y_{min} is the minimum of Y(1), Y(|K|+1), Y(2|K|+1), ..., 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 FO-F3.

TEMPORARY STORAGE

None.

LENGTH OF ROUTINE

About 81 words excluding other subroutines used.

OTHER SUBROUTINES USED

J5-CCP1PP-00-UI-AL

EXECUTION TIME

3.68 seconds for 2,000 data points.

ENTRY

The NICAP calling sequence is

CALL CCP6SC
DECQ X,Y,N,K
DECQ TX,TY,,

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

Date: 8/17/65
Section: 8.4-CCP6LN
Page: 1 of 2
Change: 0
ILLIAC II MANUAL

UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
D1-GQUL-00-UI-AL

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 μ sec for $n = 2$ to 875 μ sec for $n = 15$.
($50n + 100$) μ 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.

Programmed by: L. Lunde
Approved by:

N. T. Hamilton

Date: 11/19/63
Section: 8.4-D1-GQUL
Page: 1 of 9
Change:

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

$$\frac{q-p}{2} \sum_{i=1}^n a_i f(x_i \frac{q-p}{2} + \frac{q+p}{2}) .]$$

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 GQJ1

```

Control returned → here DECQ 10,250,, n = 10, the auxiliary subroutine f is located beginning at 250

Date:	11/19/63
Section:	8.4-D1-GQJ1
Page:	2 of 9
Change:	

MATHEMATICAL DESCRIPTION: The Gaussian quadrature formula for evaluating an integral with arbitrary limits (p,q) is given by

$$\int_p^q f(x) d(x) = \frac{q-p}{2} \sum_{i=1}^n a_i f(x_i \frac{q-p}{2} + \frac{q+p}{2}) + R_n(f)$$

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

$$R_n = \frac{f^{(2n)}(\xi)}{(2n)! K_n^2} \text{ where } \xi \text{ is a point in the interval (p,q)}$$

and

$$K_n \text{ is } \frac{\binom{2n}{n} \binom{2n+1}{n+1}^{1/2}}{(q-p)^{n+1/2}}$$

R_n is the error term. The zeros of the Legendre polynomials $P_n(x)$, $n = 2, 3, \dots, 16$, 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.

Date:	11/19/63
Section:	8.4-D1-GQU1
Page:	3 of 9
Change:	

GQU1	FIL			GQU10000
	STR	F3		GQU10001
	ADM	2,14		GQU10002
	ATN	2,1,		GQU10003
	SFR	4		GQU10004
	ATN	2,1,		GQU10005
	SFR	5		GQU10006
	ATN	2,1,		GQU10007
	SFR	6		GQU10008
	ATN	2,1,		GQU10009
	SFR	7		GQU10010
	LFR	5,M3	PICK UP PARAMETERS N AND F	GQU10011
	CAD	F3		GQU10012
	SUB	F2		GQU10013
	MPY	10,3,2048		GQU10014
	STR	M2-6	(Q-P)/2	GQU10015
	ADD	F2		GQU10016
	STR	M2-5	(Q+P)/2	GQU10017
	CRM	4,1		GQU10018
	JNM	4,GQU1+38	JUMP IF N IS ODD	GQU10019
	CRM	4,12		GQU10020
	CSB	M4.		GQU10021
	MPY	10,3,2048		GQU10022
	SIA	8		GQU10023
	CAM	1		GQU10024
	CAD	M4.		GQU10025
	SUB	2.	LOCATE XI IN TABLE	GQU10026
	TZ	GQU1+11		GQU10027
	CJU	1,2,GQU1+9		GQU10028
	CAD	M1.		GQU10029
	MPY	M1+1.		GQU10030
	ADD	GQU1+59.		GQU10031
	SIA	0		GQU10032
	ADD	M1+64.	LOCATE AI IN TABLE	GQU10033
	SIA	12		GQU10034

Date:	11/19/63
Section:	8.4-DI-GQU1
Page:	4 of 9
Change:	

CAM 9,1
 CAM 6,M2-14
 CAD M0
 MPY M2-6
 ADD M2-5
 SFR 4,M2-18
 SFR 5,M2-17
 SFR 6,M2-16
 SFR 7,M2-15
 JSB 3,M5
 FIL
 LFR 4,M2-18
 STR M6
 CSB 0,1,
 MPY M2-6
 ADD M2-5
 SFR 4,M2-18
 JSB 3,M5
 FIL
 LFR 4,M2-18
 LFR 5,M2-17
 LFR 6,M2-16
 LFR 7,M2-15
 ADD M6
 MPY 12,1,
 STR 6,1,
 SBM 9,1
 CJU 8,2,GQU1+15
 CSB M4.
 ADD 2.
 TZ GQU1+37
 CAM 7,M2-14
 CAD 7,1,
 ADD 7,1,
 CJU 9,2,GQU1+32

FORM ARGUMENT T+R

EVALUATE F(T+R)

EVALUATE F(-T+R)

FORM SUMMATION

GQU10035
 GQU10036
 GQU10037
 GQU10038
 GQU10039
 GQU10040
 GQU10041
 GQU10042
 GQU10043
 GQU10044
 GQU10045
 GQU10046
 GQU10047
 GQU10048
 GQU10049
 GQU10050
 GQU10051
 GQU10052
 GQU10053
 GQU10054
 GQU10055
 GQU10056
 GQU10057
 GQU10058
 GQU10059
 GQU10060
 GQU10061
 GQU10062
 GQU10063
 GQU10064
 GQU10065
 GQU10066
 GQU10067
 GQU10068
 GQU10069

Date: 11/19/63
 Section: 8.4-DI-GQU1
 Page: 5 of 9
 Change:

MPY M2-6
 LFR 4,M2-4
 ATN 2,1,
 LFR 5
 ATN 2,1,
 LFR 6
 ATN 2,1,
 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

GQU10070
 GQU10071
 GQU10072
 GQU10073
 GQU10074
 GQU10075
 GQU10076
 GQU10077
 GQU10078
 GQU10079
 GQU10080
 GQU10081
 GQU10082
 GQU10083
 GQU10084
 GQU10085
 GQU10086
 GQU10087
 GQU10088
 GQU10089
 GQU10090
 GQU10091
 GQU10092
 GQU10093
 GQU10094
 GQU10095
 GQU10096
 GQU10097
 GQU10098
 GQU10099
 GQU10100
 GQU10101
 GQU10102
 GQU10103
 GQU10104

Date: 11/19/63
 Section: 8.4-DI-GQU1
 Page: 6 of 9
 Change:

SFR 7,M2-15
 CAD M2-5
 JSB 3,M5
 FIL
 LFR 4,M2-18
 LFR 5,M2-17
 LFR 6,M2-16
 LFR 7,M2-15
 ATN 12,1,
 MPY 0
 CAM 6,M2-14
 STR 6,1,
 TRA 2,GQU1+15
 FIL

EVALUATE F((Q+P)/2)

OCTQ 04474,15164,05440,06400,06144,13757,04176,16200,02560
 OCTQ 10776,14735,03600,06707,03333,14754,07400,04235,11077
 OCTQ 15463,06600,07177,13316,00246,07600,07505,10455,00154
 OCTQ 16777,05224,05011,06253,11000,07353,06244,16401,04200
 OCTQ 03176,05357,10432,00000,05735,04771,13707,04000,07457
 OCTQ 10570,16370,14000,05675,06224,17170,16577,04150,11225
 OCTQ 10145,00600,06277,02253,02223,16400,07535,05434,13025
 OCTQ 11600,02460,02213,07110,12600,04720,05722,00357,17600
 OCTQ 06540,06104,17032,00200,07575,11262,16576,04400,04607
 OCTQ 02407,10443,14377,03357,03000,04401,06400,05336,15622
 OCTQ 02433,14400,06727,04625,16471,06200,07625,01740,06125
 OCTQ 16200,02120,00617,07753,04000,04116,03367,14233,01400
 OCTQ 05656,13173,15103,02600,07061,06421,12740,11200,07646
 OCTQ 15142,17657,15400,04003,15137,10765,06377,02742,12151
 OCTQ 00250,13000,04545,12360,05600,17000,06121,10256,16661
 OCTQ 01400,07167,04164,17767,01000,07664,07435,10710,05600
 OCTQ 07277,15207,11036,05377,03455,00327,10214,07200,05107
 OCTQ 01003,03540,03400,06323,04161,10470,17000,07265,07565
 OCTQ 01341,11400,07677,03327,06122,14600,03352,05751,13336
 OCTQ 07577,02433,01262,04422,15200,04076,07253,05235,16000
 OCTQ 05377,02333,00474,01600,06474,03356,04333,13200,07332

GQU10105
 GQU10106
 GQU10107
 GQU10108
 GQU10109
 GQU10110
 GQU10111
 GQU10112
 GQU10113
 GQU10114
 GQU10115
 GQU10116
 GQU10117
 GQU10118
 GQU10119
 GQU10120
 GQU10121
 GQU10122
 GQU10123
 GQU10124
 GQU10125
 GQU10126
 GQU10127
 GQU10128
 GQU10129
 GQU10130
 GQU10131
 GQU10132
 GQU10133
 GQU10134
 GQU10135
 GQU10136
 GQU10137
 GQU10138
 GQU10139

Date: 11/19/63
 Section: 8.4-DI-GQU1
 Page: 7 of 9
 Change:

OCTQ	15721,02563,03600,07707,15060,17673,05400,06340,05646	GQU10140
OCTQ	02736,00777,03116,07064,11127,15400,04442,13166,17021	GQU10141
OCTQ	16200,05627,03341,10065,06200,06622,04042,01777,17600	GQU10142
OCTQ	07377,01114,04761,06200,07716,15047,14145,14600,03024	GQU10143
OCTQ	12753,03166,12577,02201,07127,03102,03200,03524,00454	GQU10144
OCTQ	15103,12400,04742,15106,06653,15000,06026,02122,17207	GQU10145
OCTQ	03200,06731,12003,05051,14400,07434,17526,06055,17400	GQU10146
OCTQ	07724,11302,06232,17200	GQU10147
OCTQ	02000,00000,00000,00001,07070,16161,14343,11200,04343	GQU10148
OCTQ	10707,01616,03600,05157,02770,04030,04600,02620,15007	GQU10149
OCTQ	13747,13400,04432,02547,10465,07600,03650,07321,02656	GQU10150
OCTQ	04200,07451,14754,03334,00377,03574,11152,04625,04200	GQU10151
OCTQ	02705,12675,12571,04600,05366,17540,01705,13577,03257	GQU10152
OCTQ	17277,00074,03000,03033,17472,07343,02600,02171,12613	GQU10153
OCTQ	10475,10400,04111,07551,17404,15577,02715,10660,65431	GQU10154
OCTQ	11200,02404,17050,05045,05400,07073,07665,03072,05377	GQU10155
OCTQ	03172,10350,02312,00177,02510,12441,17545,04400,02377	GQU10156
OCTQ	05764,14070,15200,02053,07303,16202,07200,05617,13651	GQU10157
OCTQ	13104,05377,02463,11457,14003,16777,02272,07363,10024	GQU10158
OCTQ	15400,02116,16523,15125,11600,07005,10131,15001,07777	GQU10159
OCTQ	04620,11614,10361,16177,02104,05374,05542,15777,02135	GQU10160
OCTQ	16326,04176,00400,02064,07121,03372,05400,07354,12261	GQU10161
OCTQ	04215,14377,05754,02670,15577,16177,04011,10107,15737	GQU10162
OCTQ	01377,07100,04562,06267,07376,07762,00316,06651,16177	GQU10163
OCTQ	07361,10525,14112,15577,06400,13076,06543,05177,05076	GQU10164
OCTQ	13445,10744,05777,03330,01401,05716,06177,06023,12731	GQU10165
OCTQ	15724,15776,07342,01775,01702,13577,07173,06616,05335	GQU10166
OCTQ	02777,06514,15565,13701,04377,05546,13714,14511,04777	GQU10167
OCTQ	04343,04667,00254,13777,02745,05062,06417,14177,05135	GQU10168
OCTQ	02434,10751,11776,06706,16101,05242,12177,06441,17427	GQU10169
OCTQ	11337,05177,05737,15616,02521,16377,05017,11673,16705	GQU10170
OCTQ	12377,03706,17272,07366,01177,02441,04754,05334,07177	GQU10171
OCTQ	04375,11330,13117,16776,06367,00527,05763,11377,06263	GQU10172
OCTQ	01477,01772,00577,05752,00772,07333,10577,05244,02363	GQU10173
OCTQ	00520,15777,04356,13473,04454,06377,03333,13113,01537	GQU10174

Date: 11/19/63
 Section: 8.4-DL-GQUL
 Page: 8 of 9
 Change:

OCTQ 01177,02200,16023,00265,15177,03737,07071,01172,02576
OCTQ 06037,17256,10034,11577,05657,14307,07701,10177,05323
OCTQ 07273,13264,16777,04622,17541,13624,07577,03771,16571
OCTQ 05655,03577,03027,01167,03634,14377,07757,15432,02602
OCTQ 04776,03363,07325,12267,14176

GQU10175
GQU10176
GQU10177
GQU10178
GQU10179

Date: 11/19/63
Section: 8.4-DI-GQU1
Page: 9 of 9
Change:

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
D2-RKGL-00-UI-AL

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: $50 + 290N + 4$ (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

A	M	N	
---	---	---	--

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, \dots, y_{N-1} are stored in locations M, M+1, ..., M+N-1 respectively. The initial conditions are stored here by the user. The auxiliary subroutine uses y_0, y_1, \dots, y_{N-1} (but does not alter them) to compute the k_i 's. The solutions are found here upon return from RKGL.

Programmed by: F. Schaffer
Approved by:

E. W. Gear

Date: 7/15/64
Section: 8.4-D2-RKGL
Page: 1 of 5
Change:

USE (Continued):

- (b) k_0, k_1, \dots, k_{N-1} are stored in locations $M+N, M+N+1, \dots, M+2N-1$ respectively. The $k_i = hf_i$ are computed and placed here by the auxiliary subroutine.
- (c) q_0, q_1, \dots, q_{N-1} are stored in locations $M+2N, M+2N+1, \dots, M+3N-1$ 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

$$y'_i = f_i(y_0, y_1, y_2, \dots, y_{N-1}) \quad (i = 0, 1, 2, \dots, N - 1)$$

the process used in the integration is defined by the following equations:

$$k_{ij} = hf_i(y_{0j}, y_{1j}, \dots, y_{(N-1)j})$$

$$r_{i,j+1} = B_j(k_{ij} - A_j q_{ij})$$

$$y_{i,j+1} = y_{ij} + r_{i,j+1}$$

Date:	7/15/64
Section:	8.4-D2-RKGL
Page:	2 of 5
Change:	

METHOD (Continued):

$$q_{i,j+1} = q_{ij} + 3r_{i,j+1} - C_j k_{ij}$$

with the following table of values:

j	A _j	B _j	C _j
1	2	1/2	1/2
2	1	1 - √1/2	1 - √1/2
3	1	1 + √1/2	1 + √1/2
4	2	1/6	1/2

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_{ij} '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_{1j} 's, k_{2j} 's, k_{3j} 's and k_{4j} 's. RKG does the arithmetic indicated in the r_{ij} , y_{ij} and q_{ij} 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

$$1/15\{y_h - y_{h/2}\}$$

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.

Date:	7/15/64
Section:	8.4-D2-RKGL
Page:	3 of 5
Change:	

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-KUTTU

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=YIJ+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)KI4--(1/2)KI4

ACCU= 3RIJ-BJ*KIJ

QI,J+1=QIJ+3RIJ-BJ*KIJ

RKG1 000
 RKG1 001
 RKG1 002
 RKG1 003
 RKG1 004
 RKG1 005
 RKG1 006
 RKG1 007
 RKG1 008
 RKG1 009
 RKG1 010
 RKG1 011
 RKG1 012
 RKG1 013
 RKG1 014
 RKG1 015
 RKG1 016
 RKG1 017
 RKG1 018
 RKG1 019
 RKG1 020
 RKG1 021
 RKG1 022
 RKG1 023
 RKG1 024
 RKG1 025
 RKG1 026
 RKG1 027
 RKG1 028
 RKG1 029
 RKG1 030
 RKG1 031

Date: 7/15/64
 Section: 8.4-D2-RKG1
 Page: 4 of 5
 Change:

CJZ	11,1,RKG1+13	COUNT AND JUMP IF I= N-1	RKG1 032
SBM	10,2	RESET M10	RKG1 033
TRA	2,RKG1+5	JUMP TO CALCULATE YI+1,J+1,QI+1,J+1	RKG1 034
SBM	13,M14	RESET M13=A	RKG1 035
CJU	9,3,RKG1+3	COUNT AND JUMP IF J NOT 4	RKG1 036
SBM	2,3		RKG1 037
ATN	2,1,		RKG1 038
LFR	7		RKG1 039
ATN	2,1,		RKG1 040
LFR	6		RKG1 041
ATN	2,1,		RKG1 042
LFR	4		RKG1 043
SBM	2,3		RKG1 044
JLH	3,0,		RKG1 045
FIL			RKG1 046
OCTQ	4000,,,1,4000,,,		RKG1 047
OCTQ	2000,,,1,453,16606,6300,6201		RKG1 048
OCTQ	2000,,,1		RKG1 049
OCTQ	3324,1171,11477,11601,4000,,,1		RKG1 050
OCTQ	5252,12525,5252,12577		RKG1 051

Date:	7/15/64
Section:	8.4-D2-RKG1
Page:	5 of 5
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
E1-DVDF1-00-UI-AL

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 * 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:

f	address of auxiliary
A	address of first abscissa
k	kth divided difference

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: Jurg Nievergelt
Approved by:

lw gear

Date: 7/20/64
Section: 8.4-E1-DVDF1
Page: 1 of 4
Change:

USE:

The user must provide:

- 1) $k + 1$ abscissas X_0, X_1, \dots, X_k
in locations $A, A+1, \dots, A + k$
- 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:

k th divided difference $f[x_0, \dots, x_k]$ in location $(M2) + 2$
 $(k-1)$ st divided difference $f[x_1, \dots, x_k]$ in location $(M2) + 3$
 $(k-2)$ nd divided difference $f[x_2, \dots, x_k]$ 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 k th divided difference $f[x_0, x_1, \dots, x_k]$ 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.

Date:	7/20/64
Section:	8.4-E1-DVDF1
Page:	2 of 4
Change:	

DVDF1	ATN	2,1,
	SFR	5
	ATN	3,1,
	LFR	5
	ATN	2,1,
	SFR	4
	ATN	M5
	CAM	0
	ATN	M6
	CSM	7,1
DIVF1	CAD	0,1,
	ATN	M4
	JSB	3,0,0
	FIL	
	STR	2,1,
	CJU	7,DIVF1
	ATN	M6
	SBM	2,1
	ATN	M6
	CSM	3
DIVF2	CAM	0
	ATN	M3
	ATN	M6
	CAM	1,1
DIVF3	ATN	M5
	CSB	M0
	ATN	M5
	ADD	1,1,
	STR	F2
	ATN	M2
	CSB	0,1,
	ATN	M2

DIVIDED DIFFERENCES FIRST CARD
 SAVE F5
 INCREASE LINK BY 1
 READ PARAMETERS INTO F5

SAVE F4

M0 = X0

M7 = -(K+1)
 ABSCISSA

AUXILIARY

FILL UP S0, S1, ..., SK

M3 = -K

M1 COUNTS UP TO K FOR INNER LOOP

CURRENT ABSCISSA

NEXT ABSCISSA

STORE DIFFERENCE OF ABSCISSAS IN F2

CURRENT DIVIDED DIFFERENCE

DIVF01
 DIVF02
 DIVF03
 DIVF04
 DIVF05
 DIVF06
 DIVF07
 DIVF08
 DIVF09
 DIVF10
 DIVF11
 DIVF12
 DIVF13
 DIVF14
 DIVF15
 DIVF16
 DIVF17
 DIVF18
 DIVF19
 DIVF20
 DIVF21
 DIVF22
 DIVF23
 DIVF24
 DIVF25
 DIVF26
 DIVF27
 DIVF28
 DIVF29
 DIVF30
 DIVF31

Date: 7/20/64
 Section: 8.4-EL-DVDF1
 Page: 3 of 4
 Change:

ADD M0
DIV F2
SBM 0,3,1
STR M2
SFN M1
ATN M6
CAM 7
JPM 7,DIVF3
CJU 3,DIVF2
ATN M2
LFR 4,-1
SBM 2,2
ATN M2
LFR 5
JLH M3

NEXT DIVIDED DIFFERENCE
NEW DIVIDED DIFFERENCE

INNER LOOP
OUTER LOOP

RESTORE F4

RESTORE F5

DIVF32
DIVF33
DIVF34
DIVF35
DIVF36
DIVF37
DIVF38
DIVF39
DIVF40
DIVF41
DIVF42
DIVF43
DIVF44
DIVF45
DIVF46

Date: 7/20/64
Section: 8.4-EL-DVDF1
Page: 4 of 4
Change:

UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
ILLIAC II LIBRARY PROGRAM

E1-LAG6-00-UI-AL

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 M0 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 $C \neq 0$, there is another control word specifying another table in the next higher addressed location.

Programmed by: *C. W. Year*
Approved by:

Date: 11/15/63
Section: 8.4-E1-LAG6
Page: 1 of 5
Change:

EXAMPLE:

Suppose locations 100 to 199 contain

$$\sin \left(\frac{n\pi}{400} \right) \quad n = 0, 1, \dots, 99$$

and locations 300 to 399 contain

$$\tan \left(\frac{n\pi}{400} \right) \quad n = 0, 1, \dots, 99$$

To find $\sin \pi x$ and $\tan \pi x$, and to store in locations B and B + 1 where x is in the accumulator, the program below can be used.

```
MPY    400
CAM    1, A
CALL   LAG6
-----
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 $|f| < \frac{1}{120} 4^{63}$ 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," Columbia University Press. New York (1944), pp. xiii-xx. 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.

Date:	11/15/63
Section:	8.4-E1-LAG6
Page:	2 of 5
Change:	

ACCURACY (continued):

3. Error due to errors in table entries. This is bounded by $89/64 \max |\epsilon_i|$ where ϵ_i are the errors in the six entries used.
4. Error due to error ϵ 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 $x_0, x_0 + h, x_0 + 2h, \dots, x_0 + nh$, 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 ,

$$T_{ji} = \text{contents of location } (B_j + S_j(q - i))$$

is found, and $\sum_{i=-2}^{+3} T_{ji} A_i$ is stored in location R_j where q is the integer part of x .

Date:	11/15/63
Section:	8.4-E1-LAG6
Page:	3 of 5
Change:	

LAG6	FIL				LAG60000
	SFR	5,COMMON+6	SAVE F5		LAG60001
	SFR	6,COMMON+7	SAVE F6		LAG60002
	SIA	4	INTEGER PART Q TO M4		LAG60003
	SUB	M4.			LAG60004
	STR	F2	FRACTIONAL PART P TO F2		LAG60005
	SUB	3.			LAG60006
	STR	F3	P-3		LAG60007
	MPY	5.	-5(P-3)		LAG60008
	STN	COMMON+4			LAG60009
	CSB	2.			LAG60010
	ADD	F2			LAG60011
	MPY	F3			LAG60012
	STR	F3			LAG60013
	MPY	10.	10(P-3)(P-2)		LAG60014
	STR	COMMON+3			LAG60015
	CSB	1.			LAG60016
	ADD	F2			LAG60017
	MPY	F3			LAG60018
	STR	F3			LAG60019
	MPY	10.	-10(P-3)P-2)(P-1)		LAG60020
	STN	COMMON+2			LAG60021
	CAD	F2			LAG60022
	MPY	F3			LAG60023
	STR	F3			LAG60024
	MPY	5.	5(P-3)(P-2)(P-1)P		LAG60025
	STR	COMMON+1			LAG60026
	CAD	1.			LAG60027
	ADD	F2			LAG60028
	MPY	F3			LAG60029
	STN	COMMON	-(P-3)(P-2)P-1)P(P+1)		LAG60030
	CAN	6,-4			LAG60031
	CAN	5,COMMON+1			LAG60032
	CAD	2.			LAG60033
	ADD	F2			LAG60034

Date: 11/15/63
 Section: 8.4-E1-LAG6
 Page: 4 of 5
 Change:

STR	F3	(LAG6+13,2)	LAG60035
MPY	M5		LAG60036
STR	5,1,	A(I), I=-2,-1,0,0,0,+2	LAG60037
CAD	-3-M6.		LAG60038
ADD	F2		LAG60039
MPY	F3		LAG60040
CJU	6,2,LAG6+13		LAG60041
STR	COMMON+5	A(3)	LAG60042
CAM	7,M4	M7=Q	LAG60043
ATN	1,1,		LAG60044
LFR	6	GET NEXT CONTROL WORD	LAG60045
CAD	M9.		LAG60046
MPY	-2+M7.		LAG60047
SIA	4		LAG60048
ADM	8,M4	(Q-2)*S+B	LAG60049
CAM	6,-6		LAG60050
CAD	15,3,		LAG60051
STR	F3		LAG60052
CAM	5,COMMON		LAG60053
CAD	5,1,	(LAG6+21,1)	LAG60054
MPY	M8		LAG60055
ADM	8,M9	A-3 T-3+0,0,0+A2T2	LAG60056
ASC	F3		LAG60057
CJU	6,1,LAG6+21		LAG60058
ADD	F3		LAG60059
MPY	LAG6+26	* 1/120	LAG60060
STR	M10	STORE RESULT	LAG60061
JUM	11,0,LAG6+17	TEST FOR MORE TABLES	LAG60062
LFR	5,COMMON+6		LAG60063
LFR	6,COMMON+7		LAG60064
JLH	M3		LAG60065
FIL			LAG60066
OCTQ	4210,10421,1042,2375	1/120	LAG60067

Date: 11/15/63
 Section: 8.4-EL-LAG6
 Page: 5 of 5
 Change:

UNIVERSITY OF ILLINOIS
DIGITAL COMPUTER LABORATORY
ILLIAC II LIBRARY PROGRAM

E1-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 $B_x, B_x + S_x, \dots, B_x + (N - 1)S_x$ 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

$$T_0 = f(x_0), T_1 = f(x_1), \dots, T_{N-1} = f(x_{N-1})$$

Programmed by: <i>C. W. Gear</i> Approved by:
--

Date: 11/15/63 Section: 8.4-E1-LGUN Page: 1 of 7 Change:

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 ≠ 0, there is another control word for another tabulated function in the next higher addressed location.

EXIT:

Standard by JLH 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 A₀, A₁, ..., A_{N-1} respectively.

METHOD:

The N Lagrange coefficients A₀, A₁, ..., A_{N-1} are calculated, where

$$A_j = \prod_{\substack{i=0 \\ i \neq j}}^{N-1} \frac{(x - x_i)}{(x_j - x_i)}$$

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.

Date:	11/15/63
Section:	8.4-E1-LGUN
Page:	2 of 7
Change:	

ACCURACY (continued):

2. Truncation error in the Lagrange formula. Refer to "Tables of Lagrangian Interpolation Coefficients," Columbia University Press. New York (1944)
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

$$(4^{-64}) \frac{1}{N-1}, \quad \left(\frac{4^{-63}}{(N-1)!} \right) \frac{1}{N-1}$$

The intermediate scalar products $\sum A_i \pi$ 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, \dots, x_{99} in locations 100-199, that x_0, x_1, \dots, x_{99} are in locations 200-299

Date:	11/15/63
Section:	8.4-E1-LGUN
Page:	3 of 7
Change:	

EXAMPLE (Continued):

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.

	CAM	1, A
	CALL	LGUN
	----	----
A	DECQ	N + 98, 1, 7, COMMON + 2
	DECQ	N + 198, 1, X, 1
	DECQ	N + 298, 1, G, 0

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 $2 \leq N < 97$, so that the points used are inside the table.

Date:	11/15/63
Section:	8.4-E1-LGUN
Page:	4 of 7
Change:	

LGUN	FIL			LGUN0000
	SFR	5, COMMON	SAVE F5	LGUN0001
	SFR	6, COMMON+1	SAVE F6	LGUN0002
	STR	F2	F2=X	LGUN0003
	ATN	1, 1,		LGUN0004
	LFR	6	FIRST CONTROL WORD	LGUN0005
	SUB	M8	X-X0	LGUN0006
	CAM	4, 2-M10	M4=-N+2	LGUN0007
	CAM	5, 1+M11	M5=T+1	LGUN0008
	CAM	6, M8	M6=B	LGUN0009
	STR	5, 1,		LGUN0010
	ADM	6, M9		LGUN0011
	CAD	F2		LGUN0012
	SUB	M6		LGUN0013
	MPY	F0	(X-X(0))... (X-X(N-2))	LGUN0014
	CJU	4, LGUN+4		LGUN0015
	STR	M5		LGUN0016
	ADM	6, M9		LGUN0017
	CAM	4, 2-M10		LGUN0018
	CAD	F2		LGUN0019
	SUB	M6		LGUN0020
	STR	F3		LGUN0021
	SBM	5, 1		LGUN0022
	MPY	M5		LGUN0023
	STR	M5		LGUN0024
	CAD	F2		LGUN0025
	SBM	6, M9		LGUN0026
	SUB	M6		LGUN0027
	MPY	F3	(X-X(N-1))... (X-X(1))	LGUN0028
	CJU	4, LGUN+8		LGUN0029
	STR	M11		LGUN0030
	CAM	7, M8		LGUN0031
	CSM	6, M10		LGUN0032
	CAD	M7		LGUN0033
	STR	F2	F2=X(J)	LGUN0034

Date: 11/15/63
 Section: 8.4-EL-LGUN
 Page: 5 of 7
 Change:

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 F0
 STR F0
 CJU 4, LGUN+16
 CAD F0
 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(0)
 X(J)-X(I)
 A(J)
 NEXT CONTROL WORD
 M9--(N-1)
 A(0) T(0)
 SUM A(I)T(I)

LGUN0035
 LGUN0036
 LGUN0037
 LGUN0038
 LGUN0039
 LGUN0040
 LGUN0041
 LGUN0042
 LGUN0043
 LGUN0044
 LGUN0045
 LGUN0046
 LGUN0047
 LGUN0048
 LGUN0049
 LGUN0050
 LGUN0051
 LGUN0052
 LGUN0053
 LGUN0054
 LGUN0055
 LGUN0056
 LGUN0057
 LGUN0058
 LGUN0059
 LGUN0060
 LGUN0061
 LGUN0062
 LGUN0063
 LGUN0064
 LGUN0065
 LGUN0066
 LGUN0067
 LGUN0068
 LGUN0069

Date: 11/15/63
 Section: 8.4-FL-LGUN
 Page: 6 of 7
 Change:

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

LGUN0070
LGUN0071
LGUN0072
LGUN0073
LGUN0074
LGUN0075

Date: 11/15/63
Section: 8.4-E1-LGUN
Page: 7 of 7
Change:

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
F4-SLQ1-00-UI-AL

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 SLQ1 with:
M1 = address of parameter P.

P

N	M	A	B
---	---	---	---

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 DECQ 17,4,1000,1800

and we have

Programmed by: J. Presti
Approved by:

J. W. Geo.

Date: 7/20/64
Section: 8.4-F4-SLQ1
Page: 1 of 7
Change:

ENTRY (Continued):

CAM 1,P

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.

Date:	7/20/64
Section:	8.4-F4-SLQ1
Page:	2 of 7
Change:	

SLQ1	SFR	4,COMMON+1		
	SFR	5,COMMON+2		
	SFR	6,COMMON+3		
	SFR	7,COMMON+4		
	LFR	4,M1		SLQ 8
	CAM	8,M2		SLQ 9
	CAM	9,M3		SLQ 10
	CAM	3,1-M0		SLQ 11
	CSM	4,M0		SLQ 12
	CAD	1.		SLQ 13
	STR	COMMON		
TABLE	SFR	6,COMMON+M4+M0+5		
	ADM	9,M1		SLQ 16
	ADM	8,M0		SLQ 17
	CJU	4,0, TABLE		SLQ 18
NXCOL	LFR	5,COMMON+M3+M0+4		
	ADM	4,M0+M3-1		SLQ 20
	CAM	2,M3		SLQ 21
COMP	LFR	6,COMMON+M2+M0+5		
	ADM	8,M0+M3-1		SLQ 23
	CAD	M4		SLQ 24
	DAV	M8		SLQ 25
	JDC	2,0, SKIP		SLQ 26
	SBM	4,M0+M3-1		SLQ 27
	SFR	5,COMMON+M2+M0+5		
	CAD	F6		SLQ 29
	SAM	F5		SLQ 30
SKIP	CJU	2,0,COMP		SLQ 31
	SFR	5,COMMON+M3+M0+4		
	CAM	2,M3		SLQ 33
	CAD	1.		SLQ 34
	DIV	M4		SLQ 35

Date: 7/20/64
 Section: 8.4-F4-SLQ1
 Page: 3 of 7
 Change:

	STR	F3	SLQ	36
	CAD	M4		
	MPY	COMMON		
	STR	COMMON		
NXROW	LFR	6, COMMON+M3+M0+4		
	ADM	8, 1	SLQ	40
	LFR	7, COMMON+M2+M0+5		
	ADM	12, M0+M3-1	SLQ	42
	CAD	F3	SLQ	43
	MPY	12, 1,		
	CAM	6, M3	SLQ	45
	STR	F2	SLQ	46
	FLD		SLQ	49
	CSB	8, 1,	SLQ	50
	MPY	F2	SLQ	51
	ASC	12, 1,	SLQ	52
	CJF	6, 0,	SLQ	53
	CSM	6, M1	SLQ	54
	FLD		SLQ	55
	CSB	9, 1,	SLQ	56
	MPY	F2	SLQ	57
	ASC	13, 1,	SLQ	58
	CJF	6, 0,	SLQ	59
	CJU	2, 0, NXROW	SLQ	60
	LFR	6, COMMON+M3+M0+4		
	ADM	8, 1	SLQ	62
	CAM	2, M3	SLQ	63
	FLD		SLQ	64
	CAD	M8	SLQ	65
	MPY	F3	SLQ	66
	STR	8, 1,	SLQ	67
	CJF	2, 0,	SLQ	68
	CSM	2, M1	SLQ	69
	FLD		SLQ	70
	CAD	M9	SLQ	71

Date:	7/20/64
Section:	8.4-Fl4-SLQ1
Page:	4 of 7
Change:	

	MPY	F3		SLQ	72
	STR	9,1,		SLQ	73
	CJF	2,0,		SLQ	74
	CJU	3,0,NXCOL		SLQ	77
	LFR	5,COMMON+M0+4			
	ADM	4,M0-1		SLQ	79
	CAD	1.		SLQ	80
	DIV	M4		SLQ	81
	CSM	2,M1		SLQ	82
	STR	F2		SLQ	83
	CAD	M4			
	MPY	COMMON			
	STR	COMMON			
	FLD			SLQ	84
	CAD	M5		SLQ	85
	MPY	F2		SLQ	86
	STR	5,1,		SLQ	87
	CJF	2,0,		SLQ	88
	CAM	3,1-M0		SLQ	91
BKSUB	CAM	2,M3		SLQ	92
NXTRO	LFR	6,COMMON-M2+4			
	LDM	5,COMMON-M3+5			
	ADM	8,M2-M3+1		SLQ	95
	LFR	2,M8		SLQ	96
	CAD	15,3,			
	STR	M8		SLQ	98
	CSM	6,M1		SLQ	99
	FLD			SLQ	100
	CSB	5,1,		SLQ	101
	MPY	F2		SLQ	102
	ASC	9,1,		SLQ	103
	CJF	6,0,		SLQ	104
	CJU	2,0,NXTRO		SLQ	105
	CJU	3,0,BKSUB		SLQ	106
	LDM	13,COMMON+1			

Date:	7/20/64
Section:	8.4-Flt-SLQ1
Page:	5 of 7
Change:	

	LDM	3, M13	
	CAM	2, M3	SLQ 108
	CSM	3, M0	SLQ 109
ORDTBL	LFR	5, COMMON+M3+M0+5	
	CAM	6, M2	SLQ 111
	CAM	7, M2	SLQ 112
	ADM	2, M1	SLQ 113
	SFR	5, COMMON+M3+M0+5	
	CJU	3, 0, ORDTBL	SLQ 115
	CSM	3, M0-1	SLQ 116
	CAM	13	
ORDER	LFR	5, COMMON+M3+M0+4	
	EOM	6, 3, M5	SLQ 118
	CAM	M8	SLQ 119
	JZM	8, 0, OMIT	SLQ 120
	CAM	2, M3	SLQ 121
SWITCH	LFR	6, COMMON+M2+M0+5	
	EOM	10, 3, M5	SLQ 123
	CAM	M12	SLQ 124
	JZM	12, 0, OUT	SLQ 125
	CJU	2, 0, SWITCH	SLQ 126
OUT	CAM	10, M6	SLQ 127
	SFR	6, COMMON+M2+M0+5	
	CSM	2, M1	SLQ 129
	FLD		SLQ 130
	CAD	M11	SLQ 131
	XCH	7, 1,	SLQ 132
	STR	11, 1,	SLQ 133
	CJF	2, 0,	SLQ 134
	ADM	13, 1	
OMIT	CJU	3, 0, ORDER	SLQ 135
	CAD	COMMON	
	CRM	13, 1	
	JPM	13, DETOK	
	MPY	-1.	

Date:	7/20/64
Section:	8.4-F4-SLQ1
Page:	6 of 7
Change:	

DETOK LFR 4, COMMON+1
LFR 5, COMMON+2
LFR 6, COMMON+3
LFR 7, COMMON+4
JLH M3

Date: 7/20/64
Section: 8.4-F4-SIQ1
Page: 7 of 7
Change:

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
G5-RAN1-00-UI-AL

NAME: Random Number Generator
TYPE: Closed Subroutine
TEMPORARY STORAGE: Internal
FAST REGISTERS CHANGED: Accumulator, FO, F1
EXECUTION TIME: 25 μ sec for 13-bit numbers, 60 μ sec for 52-bit numbers
DESCRIPTION: The subroutine will generator unnormalized floating-point random numbers as specified by parameters in MO and M1. Two types of results are possible. Both are entered by


CALL RAN1

- 1) MO = number of 52-bit numbers to be generated (MO > 0).
M1 = 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.
M1 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

Programmed by: G. Cooper
Approved by:



Date: 8/10/64
Section: 8.4-G5-RAN1
Page: 1 of 5
Change:

DESCRIPTION (Continued): $F_N = (F_{N-1} \cdot 5^{19}) \bmod 2^{44}$, $F_0 = 1$

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

$$e_n = (e_{n-1} + e_{n-2}) \bmod 2^{13}, \quad e_0 = 0, e_1 = 1$$

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 $M_0 = 0$, 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

¹Random Number Generator Test Procedures Applied to a Modified, Multiplicative, Congruential Generation Method, by D. K. Chow.

²Random Number Generators for ILLIAC II, by Gilbert Cooper.

Date:	8/10/64
Section:	8.4-G5-RAN1
Page:	2 of 5
Change:	

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.

Date:	8/10/64
Section:	8.4-G5-RANL
Page:	3 of 5
Change:	

	ENTRY	RAN1,RAN1A		RAN1 000
COMMON	BSS	2		RAN1 001
RAN1	SFR	5,COMMON	SAVE F4 AND F5	RAN1 002
	SFR	4,COMMON+1		RAN1 003
	LFR	5,RANP	LOAD PARAMETERS	RAN1 004
	CSM	0,M0	SET WORD COUNTER	RAN1 005
RANL	ADM	5,M4		RAN1 006
	CAM	4,M5-M4	FORM 13 BIT RANDOM NUMBER	RAN1 007
	CRM	5,12		RAN1 008
	JUM	0,ROX	JUMP IF FULL WORDS DESIRED	RAN1 009
	CAM	0,M5	OTHERWISE KEEP ONLY 13 BITS	RAN1 010
FIN	SFR	5,RANP	SAVE PARAMETERS	RAN1 011
	LFR	5,COMMON	REPLACE F5	RAN1 012
	JLH	M3	EXIT	RAN1 013
ROX	CAD	RANP+2		RAN1 014
	MPY	RANP+1	FORM 45-BIT FRACTIONAL PART	RAN1 015
	CAE	M6	PREVENT Z SET ON LATER PASS	RAN1 016
	SAL	RANP+1	SAVE FOR NEXT GENERATION	RAN1 017
	LRS	M7	CORRECT FOR BIAS IN LAST FEW BITS	RAN1 018
	SAL	F0		RAN1 019
	CAD	F0	PUT CORRECTED NO. IN ACCUM.	RAN1 020
	ANN	5,127	USE 7 BITS FOR EXPONENT	RAN1 021
	CAE	M6		RAN1 022
	ANN	5,4096	USE SIGN BIT FOR SIGN OF FRACTION	RAN1 023
	LOR	0.		RAN1 024
	SAM	1,1,	STORE IN DESIRED SPOT AND INC. COUNTER	RAN1 025
	CJU	0,RANL	END TEST	RAN1 026
	LFR	4,COMMON+1	REPLACE F4	RAN1 027
	TRA	FIN	GO TO PRE-EXIT	RAN1 028
	FIL			RAN1 029
RANP	OCTQ	0,1,,2	PARAMETERS	RAN1 030
	OCTQ	,,,200	1. (FIXED POINT)	RAN1 031
	OCTQ	261,12127,10275,1200	5**19 (FIXED POINT)	RAN1 032
RANS	DECQ	,RANP,8,RANF	PUNCH-READ PARAMTER	RAN1 033
RANF	CHR	4,8Q6*	FORMAT	RAN1 034

Date:	8/10/64
Section:	8.4-G5-RANL
Page:	4 of 5
Change:	

```

RANIA CAM 1,RANS
      JLH M3
      GO
      ENTRY RANIB,RANIC
COMMON BSS 1
RANIB SFR 4,COMMON
      CALL RANIA
      CALL PUNCH
      LFR 4,COMMON
      JLH M3
RANIC SFR 4,COMMON
      CALL RANIA
      CALL READ
      LFR 4,COMMON
      JLH M3
      GO

```

```

GET PARAMTERS FOR PUNCH-READ
EXIT

```

```

SAVE F4
GO TO PARAMETER FETCH
PUNCH FOR NEXT TIME
REPLACE F4
EXIT
SAVE F4
GO TO PARAMETER FETCH
READ CARD FROM LAST 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

```

Date: 8/10/64
 Section: 8.4-G5-RAN1
 Page: 5 of 5
 Change:

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
J6-TOPS-01-UI-AL

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.

Programmed by: M. S. Levin
W. J. Bouknight

Approved by: *WJ Bouknight*

Date: 11/14/64
Section: 8.4-J6-TOPS
Page: 1 of 39
Change: 2

USE:

Choice of Mode:

The first two bits of SR34₈ 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:	11/14/64
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

M1 = quarter word

Length: 35 words

Other subroutines used: Number 1

Use: To output a quarter word as a positive decimal number, $0 \leq n \leq 8191$, (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

M1 = 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 PTFWA

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	3 of 39
Change:	2

6. Name: Punch or Type Sexadecimal Word
M1 = address of word (except for special entry PTSWS)
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
M1 = address of word
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

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	4 of 39
Change:	2

8. Name: Punch or Type Decimal Word (with variations)

Ml = some value as specified below

Length: 131 words

Other subroutines used: Number 1

Use: 1. Ml = address of word

To output a full word from memory in floating point decimal format, set Ml 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. Ml = 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

bbb±.000000000000E+00

3 blanks 13 2

PARAM is a location in memory specifying the parameters N and L as

PARAM DECQ N,L,,

3. Ml = not used

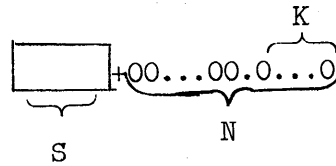
To obtain a fixed format printout of the current contents of Amost use:

CALL PTFDA

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	5 of 39
Change:	2

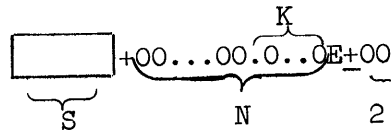
4. F specifies the format

F = 0 gives the format



This format utilizes a feature called automatic N increment (see a. in notes).

F = 1 gives the format



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.

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	7 of 39
Change:	2

Hence, N is increased until the number is $< 10^{N-K}$. 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 $10^K a_0 \geq 4^{64} = 10^{38}$, 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-DPRL-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 full word 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 full word 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)

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	9 of 39
Change:	2

Use: To dump a portion of the memory in full word
octal format with addresses, set M1 and enter via
CALL PTMDFA

12. Name: Memory Dump Control for Sexadecimal
M1 = 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 M1 and enter via

CALL PTMDS

13. Name: Memory Dump Control for Sexadecimal with
Addresses
M1 = 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 M1
and enter via

CALL PTMDSA

14. Name: Memory Dump Control for Full Word Decimal
M1 = 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 M1 and enter via

CALL PTMDD

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	10 of 39
Change:	2

15. Name: Memory Dump Control for Full Word Decimal
with Addresses
M1 = 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 M1
and enter via

CALL PFMDDA

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
M1 = 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 M1 to the address of storage,
and enter via

CALL PTFRDØ

Date:	11/14/64
Section:	8.4-J6-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
M1 = address of storage

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
M1 = address in storage

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

21. Name: Punch or Type ACCumulator Dump

M1 = not used

Length: 88 words

Other subroutines used: Number 1, Number 2, Number 6

Use: 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 contents 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 inhibited. 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.

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	13 of 39
Change:	2

TABLE 1. PTCON, Punch to Type Conversion

This is a table of the typewriter characters assigned to the various paper tape characters. Note that the following special typewriter characters are not included: $\pi \sqrt{\%}$

Decimal Value	Octal Equivalent	Paper-Tape Code	Typewriter Character Assigned	Decimal Value	Octal Equivalent	Paper-Tape Code	Typewriter Character Assigned
0	00000	0	0	64	00100))
1	00001	1	1	65	00101	#	#
2	00002	2	2	66	00102	{	[
3	00003	3	3	67	00103	}]
4	00004	4	4	68	00104	<	<
5	00005	5	5	69	00105	>	>
6	00006	6	6	70	00106	^	Δ
7	00007	7	7	71	00107	~	+
8	00010	8	8	72	00110	?	?
9	00011	9	9	73	00111	((
10	00012	+	+	74	00112	=	=
11	00013	-	-	75	00113		
12	00014	a	a	76	00114	A	A
13	00015	b	b	77	00115	B	B
14	00016	c	c	78	00116	C	C
15	00017	d	d	79	00117	D	D
16	00020	e	e	80	00120	E	E
17	00021	f	f	81	00121	F	F
18	00022	g	g	82	00122	G	G
19	00023	h	h	83	00123	H	H
20	00024	i	i	84	00124	I	I
21	00025	j	j	85	00125	J	J
22	00026	k	k	86	00126	K	K
23	00027	l	l	87	00127	L	L
24	00030	m	m	88	00130	M	M
25	00031	n	n	89	00131	N	N
26	00032	o	o	90	00132	O	O
27	00033	p	p	91	00133	P	P
28	00034	q	q	92	00134	Q	Q
29	00035	r	r	93	00135	R	R
30	00036	s	s	94	00136	S	S
31	00037	t	t	95	00137	T	T

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 14 of 39
 Change: 2

TABLE 1. PTCON, Punch to Type CONversion (Continued)

Decimal Value	Octal Equivalent	Paper-Tape Code	Typewriter Character Assigned	Decimal Value	Octal Equivalent	Paper-Tape Code	Typewriter Character Assigned
32	00040	u	u	96	00140	U	U
33	00041	v	v	97	00141	V	V
34	00042	w	w	98	00142	W	W
35	00043	x	x	99	00143	X	X
36	00044	y	y	100	00144	Y	Y
37	00045	z	z	101	00145	Z	Z
38	00046	'	'	102	00146	"	"
39	00047	:	:	103	00147	E	E
40	00050	:	:	104	00150	/	/
41	00051	.	.	105	00151	,	,
42	00052	□	NOP	106	00152		NOP
43	00053	NOP	NOP	107	00153	NOP	NOP
44	00054	NOP	NOP	108	00154	NOP	NOP
45	00055	NOP	NOP	109	00155	NOP	NOP
46	00056	NOP	NOP	110	00156	NOP	NOP
47	00057	NOP	NOP	111	00157	NOP	NOP
④48	00060	delay	S.F.O.M.	112	00160	tab	tab
49	00061	NOP	NOP	113	00161	NOP	NOP
50	00062	black	NOP	114	00162	red	NOP
51	00063	NOP	NOP	115	00163	NOP	NOP
52	00064	NOP	NOP	116	00164	NOP	NOP
53	00065	NOP	NOP	117	00165	NOP	NOP
54	00066	NOP	NOP	118	00166	NOP	NOP
55	00067	NOP	NOP	119	00167	NOP	NOP
56	00070	space	space	120	00170	backsp.	backspace
57	00071	clr tab	NOP	121	00171	set tab	NOP
58	00072	clr all	NOP	122	00172	NOP	NOP
59	00073	NOP	NOP	123	00173	NOP	NOP
60	00074	RLD	index	124	00174	HLU	NOP
61	00075	NOP	NOP	125	00175	NOP	NOP
62	00076	⊙	*	126	00176	⊙	\$
63	00077	LF/CR	LF/CR	127	00177	erase	NOP
8191	17777	special character used as the terminator for PIMSS					

④ 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

	ENTRY	PTA			TOPS	1
PTEMP	BSS	2			TOPS	2
PTCON	DECQ	4152,4217,4218,4155,4184,4121,4122,4187		01234567	TOPS	3
	DECQ	4220,4157,4128,4223,4098,4163,4196,4133		89+-ABCD	TOPS	4
	DECQ	4134,4199,4100,4165,4166,4103,4208,4145		EFGHIJKL	TOPS	5
	DECQ	4146,4211,4112,4177,4178,4115,4148,4213		MNOPQRST	TOPS	6
	DECQ	4214,4151,4180,4117,4118,4183,4097,4348		UVWXYZ	TOPS	7
	DECQ	4286,4158,,,,,		.	TOPS	8
	DECQ	3840,,,,,			TOPS	9
	DECQ	256,,,1024,,4194,768		*	TOPS	10
	DECQ	4259,4250,4285,4280,4345,4346,4249,4321)	TOPS	11
	DECQ	4351,4322,4193,4283,4226,4291,4324,4261		(= ABCD	TOPS	12
	DECQ	4262,4327,4228,4293,4294,4231,4336,4273		EFGHIJKL	TOPS	13
	DECQ	4274,4339,4240,4305,4306,4243,4276,4341		MNOPQRST	TOPS	14
	DECQ	4342,4279,4308,4245,4246,4311,4225,4262		UVWXYZ E	TOPS	15
	DECQ	4131,4160,,,,,		/,	TOPS	16
	DECQ	1280,,,,,			TOPS	17
	DECQ	512,,,,,4256,		\$	TOPS	18
PTA	SFR	4,PTEMP	FREE F4		TOPS	19
	ASN	28	READ SR34		TOPS	20
	CAM	2			TOPS	21
	JNM	2,PTA1	BYPASS OUTPUT		TOPS	22
	CRM	2,12			TOPS	23
	JNM	2,PTA2	PUNCH MODE		TOPS	24
	SFR	7,PTEMP+1	FREE F7		TOPS	25
	CRN	1,2	FETCH PROPER WORD OF PTCON		TOPS	26
	CAM				TOPS	27
	ANN	,31			TOPS	28
	LFR	7,PTCON			TOPS	29

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	16 of 39
Change:	2

	ORB	M1	TYPE PROPER CHARACTER	TOPS	30
	ATN	M12		TOPS	31
	SSR	6		TOPS	32
	LFR	7,PTEMP+1	RESTORE F7	TOPS	33
PTA1	LFR	4,PTEMP	RESTORE F4	TOPS	34
	JLH	M3	EXIT	TOPS	35
PTA2	ATN	M1	PUNCH CHARACTER	TOPS	36
	SSR			TOPS	37
	TRA	PTA1	GO TO EXIT	TOPS	38
	GO			TOPS	39

*** / NUMBER 2) PUNCH OR TYPE MESSAGE

	ENTRY	PTMSS		TOPS	40
PTEMP1	BSS	2		TOPS	41
PTMSS	SFR	4,PTEMP1	FREE F4	TOPS	42
	SFR	7,PTEMP1+1	FREE F7	TOPS	43
	CAM	,M1	SAVE ADDRESS	TOPS	44
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS	45
	CNM	2	SET LOOP COUNTER	TOPS	46
	TRA	PTMSS3	ENTER LOOP	TOPS	47
PTMSS1	CSM	2,4	SET LOOP COUNTER	TOPS	48
	ATN	,1,	FETCH WORD AND INCREMENT	TOPS	49
	LFR	7		TOPS	50
PTMSS2	ORB	M2	TEST NEXT CHARACTER FOR END	TOPS	51
	CAM	1,M12+1		TOPS	52
	JZM	1,PTMSS4	END OF MESSAGE	TOPS	53
	SBM	1,1	RESTORE CHARACTER	TOPS	54
PTMSS3	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS	55
	CJU	2,PTMSS2	MORE TO GO IN THIS WORD	TOPS	56
	TRA	PTMSS1	RETURN FOR NEW WORD	TOPS	57
PTMSS4	LFR	7,PTEMP1+1	RESTORE F7	TOPS	58
	LFR	4,PTEMP1	RESTORE F4	TOPS	59
	JLH	M3	EXIT	TOPS	60
	GO			TOPS	61

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	17 of 39
Change:	2

*** NUMBER 3A) PUNCH OR TYPE QUARTER WORD IN OCTAL

	ENTRY	PTFW,PTFW1,PTFW2,PTFW3,PTQW,PTQW1,PTQW2,PTQW3		TOPS	62
PTEMP2	BSS	3		TOPS	63
PTQW	SFR	4,PTEMP2+2	FREE F4	TOPS	64
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS	65
	TRA	PTQW5		TOPS	66
	CAN	0,0	NEEDED FOR SPACING	TOPS	67
PTQW1	SFR	4,PTEMP2+2	FREE F4	TOPS	68
	CAM	2,63	SET UP 'LF/CR'	TOPS	69
	TRA	PTQW4		TOPS	70
	CAN	0,0	NEEDED FOR SPACING	TOPS	71
PTQW2	SFR	4,PTEMP2+2	FREE F4	TOPS	72
	CAM	2,56	SET UP 'SPACE'	TOPS	73
	TRA	PTQW4		TOPS	74
	CAN	0,0	NEEDED FOR SPACING	TOPS	75
PTQW3	SFR	4,PTEMP2+2	FREE F4	TOPS	76
	CAM	2,112	SET UP 'TAB'	TOPS	77
PTQW4	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS	78
	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS	79
PTQW5	CAM	1,M2	FETCH NEXT CHARACTER	TOPS	80
	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS	81
	LDM	1,PTEMP2+2	FETCH QUARTER-WORD	TOPS	82
	CRN	1,12	LOAD SHIFTED QUARTER-WORD INTO M2	TOPS	83
	CAM	2		TOPS	84
	ANN	2,1	FETCH SIGN BIT	TOPS	85
	CAM	1		TOPS	86
	CSM	,5	SET LOOP COUNTER	TOPS	87
TRA	PTQW7	ENTER LOOP	TOPS	88	

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 18 of 39
 Change: 2

PTQW6	CRM	2,10	SHIFT FOR NEXT CHARACTER	TOPS	89
	ANN	2,7	LOAD CHARACTER INTO M1	TOPS	90
	CAM	1		TOPS	91
PTQW7	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS	92
	CJU	,PTQW6	MORE CHARACTERS	TOPS	93
	LFR	4,PTEMP2+2	RESTORE F4	TOPS	94
	JLH	M3	EXIT	TOPS	95

*** NUMBER 3B) PUNCH OR TYPE FULL WORD IN OCTAL

PTFW	SFR	4,PTEMP2	FREE F4	TOPS	96
	CAM	2	SET RELATIVIZER FOR BLANK	TOPS	97
	TRA	PTFW4		TOPS	98
PTFW1	SFR	4,PTEMP2	FREE F4	TOPS	99
	CAM	2,2	SET RELATIVIZER FOR 'LF/CR'	TOPS	100
	TRA	PTFW4		TOPS	101
PTFW2	SFR	4,PTEMP2	FREE F4	TOPS	102
	CAM	2,4	SET RELATIVIZER FOR 'SPACE'	TOPS	103
	TRA	PTFW4		TOPS	104
PTFW3	SFR	4,PTEMP2	FREE F4	TOPS	105
	CAM	2,6	SET RELATIVIZER FOR 'TAB'	TOPS	106
PTFW4	SFR	7,PTEMP2+1	FREE F7	TOPS	107
	LFR	7,M1	FETCH WORD	TOPS	108
	CSM	,4	SET QUARTER-WORD COUNTER	TOPS	109
PTFW5	ORB	M0	LOAD PROPER QUARTER-WORD	TOPS	110
	CAM	1,M12		TOPS	111
	ATN	M2	GO TO OUTPUT IT	TOPS	112
	CALL	PTQW		TOPS	113
	CAM	2,4	SET RELATIVIZER FOR 'SPACE'	TOPS	114
	CJU	,PTFW5	MORE QUARTER-WORDS TO GO	TOPS	115
	LFR	7,PTEMP2+1	RESTORE F7	TOPS	116
	LFR	4,PTEMP2	RESTORE F4	TOPS	117
	JLH	M3	EXIT	TOPS	118
	GO			TOPS	119

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	19 of 39
Change:	2

*** NUMBER 4) PUNCH DR TYPE DECIMAL QUARTER WORD

	ENTRY	PTDQ,PTDQ1,PTDQ2,PTDQ3		TOPS 120
PTEMP3	BSS	3		TOPS 121
PZERO	DECQ	???		TOPS 122
PTDQ	SFR	4,PTEMP3	SAVE F4	TOPS 123
	CAM	1,48	SET FOR OUTPUT	TOPS 124
	TRA	PTDQY		TOPS 125
PTDQ1	SFR	4,PTEMP3	SAVE F4	TOPS 126
	CAM	2,63	OUTPUT LF/CR	TOPS 127
	TRA	PTDQZ	CHARACTER	TOPS 128
PTDQ2	SFR	4,PTEMP3	SAVE F4	TOPS 129
	CAM	2,56	OUTPUT SPACE	TOPS 130
	TRA	PTDQZ	CHARACTER	TOPS 131
PTDQ3	SFR	4,PTEMP3	SAVE F4	TOPS 132
	CAM	2,112	OUTPUT TAB CHARACTER	TOPS 133
	PTDQZ	CAM	1,48	SET FOR OUTPUT FOR
PTDQZ	CALL	PTA	PREFIX CHARACTERS	TOPS 135
	CAM	1,M2		TOPS 136
	PTDQY	CALL	PTA	OUTPUT PREFIX CHARACTERS
	SFR	7,PTEMP3+1	SAVE F7	TOPS 138
	LDM	1,PTEMP3	FETCH QUARTER-WORD (N)	TOPS 139
	LFR	7,PZERO	SET ALL COUNTERS TO ZERO	TOPS 140
	JPM	1,PTDQA	N L.T. 4096	TOPS 141
	SBM	1,4100	REDUCE N	TOPS 142
	JPM	1,PTDQ4	N G. T. 4099	TOPS 143
	CAM	12,4	SET 409X TO BE OUTPUT	TOPS 144
	CAM	14,9		TOPS 145
	CAM	15,M1+10		TOPS 146
	TRA	PTDQK		TOPS 147

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 20 of 39
 Change: 2

PTDQ4	CAM	12,4	CORRECT FOR N G. T. 4099	TOPS 148
	CAM	13,1		TOPS 149
PTDQA	SBM	1,1000	EXTRACT	TOPS 150
	JNM	1,PTDQB	THOUSANDS	TOPS 151
	CJU	12,PTDQA	DIGIT	TOPS 152
PTDQB	ADM	1,1000		TOPS 153
PTDQC	SBM	1,100	EXTRACT	TOPS 154
	JNM	1,PTDQD	HUNDREDS	TOPS 155
	CJU	13,PTDQC	DIGIT	TOPS 156
PTDQD	ADM	1,100		TOPS 157
PTDQE	SBM	1,10	EXTRACT	TOPS 158
	JNM	1,PTDQF	TENS	TOPS 159
	CJU	14,PTDQE	DIGIT	TOPS 160
PTDQF	ADM	15,M1+10	EXTRACT UNITS DIGIT	TOPS 161
	SBN	13,10	CORRECT FOR CARRY	TOPS 162
	CAM	1		TOPS 163
	JUM	1,PTDQ5		TOPS 164
	CAM	13		TOPS 165
	CJU	12,PTDQK		TOPS 166
PTDQ5	JUM	12,PTDQK	SCAN TO ELIMINATE	TOPS 167
	CAM	12,56	LEADING ZEROS	TOPS 168
	JUM	13,PTDQK		TOPS 169
	CAM	13,56		TOPS 170
	JUM	14,PTDQK		TOPS 171
	CAM	14,56		TOPS 172
PTDQK	CSM	2,4	SET DIGIT COUNTER	TOPS 173
PTDQ6	ORB	M2	OUTPUT	TOPS 174
	CAM	1,M12	ALL	TOPS 175
	CALL	PTA	DIGITS	TOPS 176
	CJU	2,PTDQ6		TOPS 177
	LFR	7,PTEMP3+1	RESTORE F7	TOPS 178
	LFR	4,PTEMP3	RESTORE F4	TOPS 179
	JLH	M3	EXIT	TOPS 180
	GO			TOPS 181

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	21 of 39
Change:	2

*** NUMBER 5) PUNCH OR TYPE FULL WORD IN OCTAL WITH ADDRESS

	ENTRY	PTFWA		TOPS 182
PTMP2A	CALL	PTFW2	NEEDED TO SET UP TRANSFER VECTOR	TOPS 183
PTFWA	SFR	4,PTMP2A	FREE F4	TOPS 184
	CALL	PTDQ1	GO TO OUTPUT ADDRESS	TOPS 185
	CALL	PTQW2		TOPS 186
	LFR	4,PTMP2A	RESTORE F4	TOPS 187
	TRA	PTFW2	GO TO OUTPUT WORD	TOPS 188
	GO			TOPS 189

*** NUMBER 6) PUNCH OR TYPE SEXADECIMAL WORD

	ENTRY	PTSW,PTSW1,PTSW2,PTSW3,PTSW4		TOPS 190
PTEMP4	BSS	3		TOPS 191
PTSW5	SFR	4,PTEMP4	FREE F4	TOPS 192
	SFR	5,PTEMP4+1	FREE F5	TOPS 193
	SFR	7,PTEMP4+2	FREE F7	TOPS 194
	CAM	15,M1	SAVE EXPONENT	TOPS 195
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 196
	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 197
	TRA	PTSW7		TOPS 198
PTSW	SFR	4,PTEMP4	FREE F4	TOPS 199
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 200
	TRA	PTSW5		TOPS 201
PTSW1	SFR	4,PTEMP4	FREE F4	TOPS 202
	CAM	2,63	SET UP 'LF/CR'	TOPS 203
	TRA	PTSW4		TOPS 204
PTSW2	SFR	4,PTEMP4	FREE F4	TOPS 205
	CAM	2,56	SET UP 'SPACE'	TOPS 206
	TRA	PTSW4		TOPS 207
PTSW3	SFR	4,PTEMP4	FREE F4	TOPS 208
	CAM	2,112	SET UP 'TAB'	TOPS 209
PTSW4	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 210

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 22 of 39
 Change: 2

	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 211
	CAM	1,M2	FETCH NEXT CHARACTER	TOPS 212
PTSW5	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 213
	SFR	5,PTEMP4+1	FREE F5	TOPS 214
	SFR	7,PTEMP4+2	FREE F7	TOPS 215
	LDM	1,PTEMP4	FETCH ADDRESS	TOPS 216
	LFR	5,M1	LOAD WORD	TOPS 217
	CAM	15,M7	LOAD EXPONENT	TOPS 218
	CRN	15,7	EXTEND EXPONENT TO 8 BITS	TOPS 219
	CAM	14		TOPS 220
	JNM	14,PTSW6		TOPS 221
	ANM	15,127		TOPS 222
	TRA	PTSW7		TOPS 223
PTSW6	ORM	15,128		TOPS 224
PTSW7	CRM	4,12	LOAD SIGN BIT	TOPS 225
	ANN	4,1		TOPS 226
	CAM	1		TOPS 227
	CNM	12	SET GROUP COUNTERS	TOPS 228
	CSM	13,3		TOPS 229
	CSM	14,4	SET DIGIT COUNTER	TOPS 230
	TRA	PTSW10	ENTER LOOP	TOPS 231
PTSW8	CSM	14,3	SET DIGIT COUNTER	TOPS 232
PTSW9	CRM	4,9	LOAD NEXT DIGIT	TOPS 233
	ANN	4,15		TOPS 234
	CAM	1		TOPS 235
PTSW10	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 236
	CJU	14,PTSW9	MORE DIGITS TO GO	TOPS 237
	CJZ	13,PTSW11	3 GROUPS PUNCHED	TOPS 238
	CAM	4,M5	LOAD GROUP 2	TOPS 239
	CJZ	12,PTSW8	1 GROUP PUNCHED	TOPS 240
	ANM	4,1	LOAD GROUP 3	TOPS 241
	ANN	6,8190	(8190 = 17776)	TOPS 242
	ADM	4		TOPS 243
	CRM	4,1		TOPS 244

Date:	11/14/64
Section:	8.4-J6-ITOPS
Page:	23 of 39
Change:	

	TRA	PTSW8	ENTER LOOP	TOPS 245
PTSW11	ANM	6,3	LOAD GROUP 4	TOPS 246
	ANN	7,8064	(8064 = 17600)	TOPS 247
	ADM	6		TOPS 248
	CRM	6,2		TOPS 249
	CSM	14,2	SET DIGIT COUNTER	TOPS 250
PTSW12	CRM	6,9	LOAD NEXT DIGIT	TOPS 251
	ANN	6,15		TOPS 252
	CAM	1		TOPS 253
PTSW13	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 254
	CJU	14,PTSW12	MORE DIGITS TO GO	TOPS 255
	JUM	13,PTEX5	END SIGNAL SET	TOPS 256
	CAM	1,56	SET UP 'SPACE'	TOPS 257
	CRN	15,8	LOAD EXPONENT	TOPS 258
	CAM	6		TOPS 259
	CSM	14,3	SET DIGIT COUNTER	TOPS 260
	CJU	13,PTSW13	ENTER LOOP AND SET END SIGNAL	TOPS 261
PTEX5	LFR	7,PTMP4+2	RESTORE F7	TOPS 262
	LFR	5,PTMP4+1	RESTORE F5	TOPS 263
	LFR	4,PTMP4	RESTORE F4	TOPS 264
	JLH	M3	EXIT	TOPS 265
	GO			TOPS 266

*** NUMBER 7) PUNCH OR TYPE SEXADECIMAL WORD WITH ADDRESS

	ENTRY	PTSWA		TOPS 267
PTMP4A	CALL	PTSW2	NEEDED TO SET UP TRANSFER VECTOR	TOPS 268
PTSWA	SFR	4,PTMP4A	FREE F4	TOPS 269
	CALL	PTDQ1	GO TO OUTPUT ADDRESS	TOPS 270
	CALL	PTQW2		TOPS 271
	LFR	4,PTMP4A	RESTORE F4	TOPS 272
	TRA	PTSW2	GO TO OUTPUT WORD	TOPS 273
	GO			TOPS 274

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 24 of 39
 Change: 2

*** NUMBER 8) PUNCH OR TYPE DECIMAL WORD (WITH VARIATIONS)

	ENTRY	PTDW,PTDW1,PTDW2,PTDW3		TOPS 275	
	ENTRY	PTFDW,PTFDA,PTFDWA		TOPS 276	
PTEMP8	BSS	15		TOPS 277	
PTDW	SFR	4,PTEMP8+1	FREE F4	TOPS 278	
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 279	
	TRA	PTDW5		TOPS 280	
	CAN	0,0	NEEDED FOR SPACING	TOPS 281	
PTDW1	SFR	4,PTEMP8+1	FREE F4	TOPS 282	
	CAM	2,63	SET UP 'LF/CR'	TOPS 283	
	TRA	PTDW4		TOPS 284	
	CAN	0,0	NEEDED FOR SPACING	TOPS 285	
PTDW2	SFR	4,PTEMP8+1	FREE F4	TOPS 286	
	CAM	2,56	SET UP 'SPACE'	TOPS 287	
	TRA	PTDW4		TOPS 288	
	CAN	0,0	NEEDED FOR SPACING	TOPS 289	
PTDW3	SFR	4,PTEMP8+1	FREE F4	TOPS 290	
	CAM	2,112	SET UP 'TAB'	TOPS 291	
PTDW4	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 292	
	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 293	
	CAM	1,M2	FETCH NEXT CHARACTER	TOPS 294	
PTDW5	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 295	
	SFR	6,PTEMP8	SAVE F6	TOPS 296	
	SFR	7,PTEMP8+5	SAVE F7	TOPS 297	
	LDM	1,PTEMP8+1	RESTORE M1	TOPS 298	
	CSM	0,4	SET ENTRY FLAG TO 4	TOPS 299	
	TRA	REL6	PROCEED	TOPS 300	
	PTFDW	SFR	6,PTEMP8	SAVE F6	TOPS 301
		SFR	7,PTEMP8+5	SAVE F7	TOPS 302
SFR		4,PTEMP8+1	SAVE F4	TOPS 303	
CAM		1,48	SET FOR	TOPS 304	
CALL		PTA	OUTPUT MODE	TOPS 305	
	LDM	1,PTEMP8+1	RESTORE M1	TOPS 306	

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	25 of 39
Change:	2

	CSM	0,3	SET ENTRY FLAG TO 3	TOPS 307
	TRA	REL6	PROCEED	TOPS 308
PTFDA	SFR	6,PTEMP8	SAVE F6	TOPS 309
	SFR	7,PTEMP8+5	SAVE F7	TOPS 310
	SFR	4,PTEMP8+1	SAVE F4	TOPS 311
	CAM	1,48	SET FOR	TOPS 312
	CALL	PTA	OUTPUT MODE	TOPS 313
	LDM	1,PTEMP8+1	RESTORE M1	TOPS 314
	CSM	0,2	SET ENTRY FLAG TO 2	TOPS 315
	TRA	REL6	PROCEED	TOPS 316
PTFDWA	SFR	6,PTEMP8	SAVE F6	TOPS 317
	SFR	7,PTEMP8+5	SAVE F7	TOPS 318
	LFR	6,M1	LOAD PARAMETER WORD	TOPS 319
	SFR	4,PTEMP8+1	SAVE F4	TOPS 320
	CAM	1,48	SET FOR	TOPS 321
	CALL	PTA	OUTPUT MODE	TOPS 322
	LDM	1,PTEMP8+1	RESTORE M1	TOPS 323
	LFR	4,M1+1	LOAD 2ND PARAMETER WORD	TOPS 324
	CAM	13,M1	SAVE SIGN IN M13	TOPS 325
REL6	SFR	3,PTEMP8+2	SAVE F3	TOPS 326
	SFR	5,PTEMP8+3	SAVE F5	TOPS 327
	SFR	4,PTEMP8+14	SAVE F4	TOPS 328
	CAM	7,PTEMP8+8	SET TEMP STORE STARTING ADDRESS	TOPS 329
	CAM	2	RESET OV, Z FLAGS	TOPS 330
	TNOR	JB3	OV NOT SET	TOPS 331
	ADM	2,4096	SET OV FLAG	TOPS 332
JB3	TU	JB4	Z NOT ON	TOPS 333
	ADM	2,1	SET Z FLAG	TOPS 334
JB4	ATN	7,1,	SAVE F0=OUT	TOPS 335
	SFR			TOPS 336
	SEX	M3	SAVE EXPONENT	TOPS 337
	ATN	7,1,	SAVE Z,OV FLAGS	TOPS 338
	SFR	4	AND EXPONENT	TOPS 339
	SAM	7,1,	SAVE AMOST	TOPS 340

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	26 of 39
Change:	2

	SAL	7,1,	SAVE ALEAST	TOPS 341
	CAD	F1	SAVE F1=IN	TOPS 342
	SAM	M7		TOPS 343
	SFR	4,PTEMP8+7	SAVE OV, Z FLAGS	TOPS 344
	LFR	4,PTEMP8+14	RESTORE F4	TOPS 345
	ADM	0,1	JUMP IF ENTERED	TOPS 346
	JPM	0,REL14	BY PTFDWA	TOPS 347
	ADM	0,1	CHECK FOR	TOPS 348
	JZM	0,REL9B	PTFDA ENTRY	TOPS 349
	ADM	0,1	CHECK FOR	TOPS 350
	JZM	0,REL9A	PTFDW ENTRY	TOPS 351
	ADM	0,1	RESET FOR CORRECT FORMAT	TOPS 352
	CAD	M1	LOAD WORD FOR PRINTING	TOPS 353
REL9B	CAM	8	SET FOR NO PRECEEDING CHARACTER	TOPS 354
	TRA	REL11B	JUMP FOR PTFDA ENTRY	TOPS 355
REL9A	LFR	5,M1	LOAD PARAMETER WORD	TOPS 356
	SBM	0,1	RESET CORRECT FORMAT	TOPS 357
	CSM	5,M5	SET COUNTER	TOPS 358
	CSM	6,1	SET QUAD. =-1	TOPS 359
REL9	CAD	4,1,	LOAD CURRENT WORD	TOPS 360
	CJU	6,REL11A	JUMP ON WORD ON LINE COUNT	TOPS 361
	CSM	6,6	SET WORD ON LINE COUNT	TOPS 362
REL11	SFN	4	SET S=-1	TOPS 363
REL11A	CSM	8,3	SET S=3	TOPS 364
REL11B	CSM	9,13	SET N=13	TOPS 365
	CAM	10,13	SET K=13	TOPS 366
	SBM	0,1	RESET CORRECT FORMULA	TOPS 367
	CAM	13,10	SET FOR POSITIVE SIGN '+'	TOPS 368
	CAM	2	SUPPRESS LEADING ZEROS	TOPS 369
	TRA	REL15	SKIP OVER	TOPS 370
REL14	CSM	8,M8	RESET ENTRY	TOPS 371
	CSM	9,M9		TOPS 372
REL15	CAM	7	CLEAR SIGN FLAG	TOPS 373
	SFR	2,PTEMP8+4	SAVE F2	TOPS 374

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	27 of 39
Change:	2

	TZP	REL17	JUMP IF AMOST G.T.E. ZERO	TOPS 375
	STN	F3	SET NUMBER POSITIVE	TOPS 376
	CAM	13,11	SET FOR MINUS SIGN '-'	TOPS 377
REL17	STR	F3	PUT NUMBER IN F3	TOPS 378
	JNM	8,REL20	JUMP IF S G.T.E. 1	TOPS 379
	JZM	8,REL21	JUMP IF S = 0	TOPS 380
	CAM	1,63	PRINT	TOPS 381
	CALL	PTA	LF/CR	TOPS 382
	TRA	REL21	PROCEED	TOPS 383
REL20	CAM	1,56	PRINT	TOPS 384
	CALL	PTA	SPACES	TOPS 385
	CJU	8,REL20		TOPS 386
REL21	JZM	0,REL65	JUMP FOR ENTRY BY PTFDW	TOPS 387
	CAM	11,M10+M9-1	SET M1=-(N-K+1)	TOPS 388
	CAD	F3	LOAD NUMBER IN AMOST	TOPS 389
	TZ	REL32	JUMP IF NUMBER=0	TOPS 390
	CAM	8,M9+1	SET S=1-N	TOPS 391
	CAD	1.	LOAD 1.0 INTO AMOST	TOPS 392
	JZM	8,REL25	JUMP IF S=0	TOPS 393
J1	MPY	10.	MPY BY 10.0	TOPS 394
	CJU	8,J1	FORM 10**W-1)	TOPS 395
REL25	STR	F2	STORE 10**(N-1)	TOPS 396
REL26	CAD	F3	LOAD F3 IN AMOST	TOPS 397
	DAV	F2	ABS(A0)-10**(N-1) IN AMOST	TOPS 398
	TZP	REL29	ABS(A0)G.T.E. 10**(N-1)	TOPS 399
	CAD	F3	FORM	TOPS 400
	MPY	10.	10A0	TOPS 401
	STR	F3	SUBTRACT 1 FROM EXPONENT	TOPS 402
	SBM	10,1	FOR EVERY MULTIPLICATION	TOPS 403
	TRA	REL26	RETURN TO TRY AGAIN	TOPS 404
REL29	CAD	F2	10**(N-1) IN AMOST	TOPS 405
	MPY	10.		TOPS 406
	STR	F2	10**N	TOPS 407
REL30	CAD	F3	LOAD F3 IN AMOST	TOPS 408

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	28 of 39
Change:	2

	DAV	F2	ABS(A0)-10**N	TOPS 409
	TN	REL33	JUMP IF ABS(A0) L.T. 10**N	TOPS 410
	CAD	F3	LOAD F3 IN AMOST	TOPS 411
	DIV	10.	DIV BY 10.	TOPS 412
	STR	F3	SAVE IN F3	TOPS 413
	ADM	10,1	INCREASE EXPONENT COUNT	TOPS 414
	TRA	REL30	RETURN	TOPS 415
REL32	CSM	10,99	SET EXPONENT=-99 IF A0=0	TOPS 416
REL33	TOR	REL33A	CLEAR OV	TOPS 417
REL33A	CAD	F3	LOAD F3 INTO AMOST	TOPS 418
	ADD	10,3,2048	ROUND A0 AND	TOPS 419
	SIF	F2	STORE INTEGER FIX POINT IN F2	TOPS 420
	TOR	REL35	JUMP IF OV	TOPS 421
	SIF	F3	STORE INTEGER FIXED PT. IN F3	TOPS 422
REL35	CAM	8,1+M9	SET S=1-N	TOPS 423
	CAD	1.	LOAD 1 IN AMOST	TOPS 424
	JZM	8,REL38	JUMP IF S=0	TOPS 425
REL36	MPY	10.	MPY BY 10.0	TOPS 426
	CJU	8,REL36	RETURN FOR MORE	TOPS 427
REL38	STR	F2	10**N-1 TO F2	TOPS 428
	CAM	8,M0	TRUE FORMAT IN M8	TOPS 429
	CSM	15,1	SET COUNTER FOR TEST OF D=10	TOPS 430
	CAD	F3	LOAD F3 INTO AMOST	TOPS 431
REL40	CJU	9,REL40A	INCREASE M2 AFTER (N-1)	TOPS 432
	CJU	2,REL40A	TIMES THROUGH LOOP	TOPS 433
REL40A	DIV	F2	0=INTEGER PART OF	TOPS 434
	SIA	M12	A0/(10**(N-1))	TOPS 435
	CJU	11,REL41		TOPS 436
	CJU	2,REL41		TOPS 437
REL41	JZM	12,REL42	JUMP IF D=0	TOPS 438
	ADM	2,1	M14+1 IF D N.E. 0	TOPS 439
REL42	JZM	2,REL49	JUMP IF DIGITS SUPPRESSED	TOPS 440
	JUM	7,REL44	JUMP IF SIGN ALL READY PRINTED	TOPS 441
	CNM	M7	M7 N.E. 0	TOPS 442

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	29 of 39
Change:	2

	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	TOPS 443
JUMP IF POINT ALL READY PRINTED	TOPS 444
OUTPUT	TOPS 445
DECIMAL POINT	TOPS 446
JUMP EXCEPT IN FIRST PART	TOPS 447
	TOPS 448
	TOPS 449
JUMP IF D N.E. 10	TOPS 450
IF D=10, SET D=1	TOPS 451
AND INCREASE EXPONENT BY 1	TOPS 452
	TOPS 453
PRINT	TOPS 454
D	TOPS 455
	TOPS 456
JUMP IF SUPPRESSED LEADING ZEROS	TOPS 457
ARE REPLACED BY NOTHING	TOPS 458
OUTPUT	TOPS 459
SPACE	TOPS 460
FORM	TOPS 461
$A=10(A-(10*(N-1))D)$	TOPS 462
	TOPS 463
	TOPS 464
	TOPS 465
	TOPS 466
	TOPS 467
	TOPS 468
	TOPS 469
	TOPS 470
	TOPS 471
	TOPS 472
PUT EXPONENT	TOPS 473
INTO AMOST	TOPS 474
JUMP IF EXPONENT POSITIVE	TOPS 475
CHANGE SIGN	TOPS 476

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	30 of 39
Change:	2

	ATN	1	PRINT '-'	TOPS 477
REL56	CAM	1,10	PRINT '+'	TOPS 478
	CALL	PTA		TOPS 479
	STR	F3		TOPS 480
	CAM	0	CHANGE FORMAT	TOPS 481
	CSM	9,2	SET N=2	TOPS 482
	CSM	11,3	-(N-K+1)=-3	TOPS 483
	TRA	REL40		TOPS 484
REL59	LFR	7,PTEMP8+5	RESTORE F4	TOPS 485
	LFR	2,PTEMP8+4	RESTORE F2	TOPS 486
	ADM	8,1	(M8)+1	TOPS 487
	JPM	8,REL62		TOPS 488
	CSM	0,1	RESET CORRECT FORMAT	TOPS 489
	CJU	5,REL9		TOPS 490
REL62	LFR	3,PTEMP8+2		TOPS 491
	CAM	1,PTEMP8+8	SET TEMP STORE STARTING ADDRESS	TOPS 492
	CAD	1,1,	RESTORE	TOPS 493
	SAM	F0	F0=OUT	TOPS 494
	ATN	1,1,	FETCH Z,OV FLAGS	TOPS 495
	LFR	5	AND EXPONENT	TOPS 496
	JPM	6,JB5	JUMP IF OV NOT SET	TOPS 497
	CAD	13.	SET OV	TOPS 498
	DIV	15,3,		TOPS 499
JB5	CAD	1,1,	LOAD AMOST	TOPS 500
	LAL	1,1,	LOAD ALEAST	TOPS 501
	CAE	M7	RESTORE EXPONENT	TOPS 502
	CRM	6,1		TOPS 503
	JPM	6,JB6	JUMP IF Z NOT SET	TOPS 504
	ADE	-128	SET Z	TOPS 505
	ADE	-128		TOPS 506
JB6	ATN	1,1,	RESTORE	TOPS 507
	LFR	5	F1=IN	TOPS 508
	OCTQ	10727		TOPS 509
	LFR	4,PTEMP8+1	RESTORE F4	TOPS 510

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	31 of 39
Change:	2

	LFR	5, PTEMP8+3	RESTORE F5	TOPS 511
	LFR	6, PTEMP8	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**K)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**K)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			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
PTDWA	SFR	4, PTMP8A	FREE F4	TOPS 535
	CALL	PTDQ1	GO TO OUTPUT ADDRESS	TOPS 536
	CALL	PTQW2		TOPS 537
	LFR	4, PTMP8A	RESTORE F4	TOPS 538
	TRA	PTDW2	GO TO OUTPUT WORD	TOPS 539
	GO			TOPS 540

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	32 of 39
Change:	2

```

*** 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
      TOPS 541
      TOPS 542
      TOPS 543
      TOPS 544
      TOPS 545
      TOPS 546
      TOPS 547

*** 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
      TOPS 548
      TOPS 549
      TOPS 550
      TOPS 551
      TOPS 552
      TOPS 553
      TOPS 554

*** 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
      TOPS 555
      TOPS 556
      TOPS 557
      TOPS 558
      TOPS 559
      TOPS 560
      TOPS 561

*** 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
      TOPS 562
      TOPS 563
      TOPS 564
      TOPS 565
      TOPS 566
      TOPS 567
      TOPS 568

```

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	33 of 39
Change:	2

```

*** NUMBER 14) MEMORY DUMP CONTROL FOR DECIMAL WORD
ENTRY PTMDD
PTMP6E BSS 1 TOPS 569
CALL PTDW1 TOPS 570
PTMDD SFR 4,PTMP6E FREE F4 TOPS 571
CAM ,PTMP6E+1 LOAD LOCATION OF JUMP INSTRUCTION TOPS 572
CALL PTMDX TOPS 573
GO TOPS 574
TOPS 575

*** NUMBER 15) MEMORY DUMP CONTROL FOR DECIMAL WORD WITH ADDRESSES
ENTRY PTMDDA
PTMP6F BSS 1 TOPS 576
CALL PTDWA TOPS 577
PTMDDA SFR 4,PTMP6F FREE F4 TOPS 578
CAM ,PTMP6F+1 LOAD LOCATION OF JUMP INSTRUCTION TOPS 579
CALL PTMDX TOPS 580
GO TOPS 581
TOPS 582

*** NUMBER 16) PUNCH OR TYPE MEMORY DUMP
ENTRY PTMDX
PTMP6 BSS 1 TOPS 583
PTMDX SFR 5,PTMP6 FREE F5 TOPS 584
LFR 5,M0 FETCH JUMP INSTRUCTION TOPS 585
SFR 5,PTMDX1 STORE JUMP INSTRUCTION TOPS 586
LDM 2,M1 FETCH L.W.A. TOPS 587
LDM 1,M1 FETCH F.W.A. TOPS 588
CSM 2,M2-M1+1 SET WORD COUNTER TOPS 589
FIL TOPS 590
PTMDX1 JSB 3,PTMDX (JUMP TO PROPER SUBROUTINE TOPS 591
FIL IS STORED HERE) TOPS 592
ADM 1,1 INCREMENT WORD COUNT TOPS 593
CJU 2,PTMDX1 MORE WORDS TO BE DUMPED TOPS 594
LFR 5,PTMP6 RESTORE F5 TOPS 595
LFR 4,M0-1 RESTORE F4 TOPS 596
JLH M3 EXIT TOPS 597
GO TOPS 598
TOPS 599

```

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 34 of 39
 Change:

```

*** NUMBER 17) FAST REGISTER DUMP CONTROL FOR FULL WORD OCTAL
ENTRY PTFRDO
PTFRDO CAM 2,PTMP7A LOAD LOCATION OF JUMP INSTRUCTION TOPS 600
CAM 0,M3 SAVE LINK TOPS 601
CALL PTFRD TOPS 602
PTMP7A CALL PTFW2 TOPS 603
GO TOPS 604
TOPS 605

```

```

*** NUMBER 18) FAST REGISTER DUMP CONTROL FOR SEXADECIMAL WORD
ENTRY PTFRDS
PTFRDS CAM 2,PTMP7B LOAD LOCATION OF JUMP INSTRUCTION TOPS 606
CAM 0,M3 SAVE LINK TOPS 607
CALL PTFRD TOPS 608
PTMP7B CALL PTSW2 TOPS 609
GO TOPS 610
TOPS 611

```

```

*** NUMBER 19) FAST REGISTER DUMP CONTROL FOR DECIMAL WORD
ENTRY PTFRDD
PTFRDD CAM 2,PTMP7C LOAD LOCATION OF JUMP INSTRUCTION TOPS 612
CAM 0,M3 SAVE LINK TOPS 613
CALL PTFRD TOPS 614
PTMP7C CALL PTDW2 TOPS 615
GO TOPS 616
TOPS 617

```

```

*** NUMBER 20) PUNCH OR TYPE FAST REGISTER DUMP
ENTRY PTFRD
PTEMP7 BSS 7 TOPS 618
PTFRD SFR 7,PTEMP7+5 FREE F7 TOPS 619
LFR 7,M2 FETCH JUMP INSTRUCTION TOPS 620
SFR 7,PTFRDB STORE JUMP INSTRUCTION TOPS 621
LFR 7,M1 FETCH ORIGINAL F4 TOPS 622
SFR 7,PTEMP7+2 STORE ORIGINAL F4 TOPS 623
TOPS 624

```

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 35 of 39
 Change: 2

	CAM	12,2948	(2948 = 05604) SET LINK	TOPS 625
	CAM	13,M0		TOPS 626
	SFR	7,PTFRD2	STORE LINK	TOPS 627
	SFR	2,PTEMP7	STORE OTHER FAST REGISTERS	TOPS 628
	SFR	3,PTEMP7+1		TOPS 629
	SFR	5,PTEMP7+3		TOPS 630
	SFR	6,PTEMP7+4		TOPS 631
	CAM	1,48	SET UP SPECIAL CHARACTER	TOPS 632
	CALL	PTA	GO TO OUTPUT CHARACTER	TOPS 633
	CSM	0,6	SET LOOP COUNTER	TOPS 634
PTFRDA	CAM	1,63	SET UP 'LF/CR'	TOPS 635
	CALL	PTA	GO TO OUTPUT IT	TOPS 636
	CAM	1,81	SET UP 'F'	TOPS 637
	CALL	PTA	GO TO OUTPUT IT	TOPS 638
	CAM	1,M0+8	SET UP NUMBER OF FAST REGISTER	TOPS 639
	CALL	PTA	GO TO OUTPUT IT	TOPS 640
	CAM	1,PTEMP7+6+M0	SET ADDRESS OF FAST REGISTER	TOPS 641
	FIL			TOPS 642
PTFRDB	JSB	3,PTFRD	(JUMP TO PROPER SUBROUTINE	TOPS 643
	FIL		IS STORED HERE)	TOPS 644
	CJU	0,PTFRDA	MORE TO GO	TOPS 645
	LFR	4,PTEMP7+2	RESTORE F4	TOPS 646
	LFR	7,PTEMP7+5	RESTORE	TOPS 647
	FIL			TOPS 648
PTFRD2	TRA	M3	EXIT (LINK IS	TOPS 649
	FIL		STORED HERE)	TOPS 650
	GO			TOPS 651

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	36 of 39
Change:	2

*** NUMBER 21) PUNCH OR TYPE ACCUMULATOR DUMP

ENTRY	PTACC		TOPS
PTEMP5	BSS	10	652
PTAMS1	DECQ	63,76,78,78,96,88,96,87,76,95,90,93	653
	DECQ	56,79,96,88,91,63,76,24,26,30,31,56	ACCUMULATOR TOPS 654
	DECQ	56,56,8191,,56,56,56,90,97,56,20,25	DUMP AMOST TOPS 655
	DECQ	15,20,14,12,31,26,29,56,26,25,56,56	OV IN TOPS 656
PTAMS2	DECQ	63,76,23,16,12,30,31,56,56,8191,,	DICATOR ON TOPS 657
	DECQ	56,56,56,101,56,20,25,15,20,14,12,31	ALEAST TOPS 658
	DECQ	26,29,56,26,25,56,56,56	Z INDICAT TOPS 659
PTAMS3	DECQ	63,93,105,80,94,56,56,56,56,8191,,	OR ON TOPS 660
	DECQ	56,56,56,90,97,56,30,16,31,56,15,32	R,ES TOPS 661
	DECQ	29,20,25,18,56,30,31,26,29,12,18,16	OV SET DU TOPS 662
PTAMS4	DECQ	63,81,,11,90,32,31,56,56,8191,,	RING STORAGE TOPS 663
PTAMS5	DECQ	63,81,1,11,84,25,56,56,56,8191	FO-OUT TOPS 664
PTACC	SFR	4,PTEMP5+2 SAVE F4	F1-IN TOPS 665
	SFR	5,PTEMP5+3 SAVE F5	TOPS 666
	SFR	6,PTEMP5+4 SAVE F6	TOPS 667
	SFR	7,PTEMP5+5 SAVE F7	TOPS 668
	CAM	8 SET FLAGS TO ZERO	TOPS 669
	CAM	9	TOPS 670
	CAM	10	TOPS 671
	CAM	11,PTEMP5+6 SET BASE ADDRESS OF STORAGE	TOPS 672
	SFR	0,PTEMP5+1 SAVE F0	TOPS 673
	TNOR	PTACC1	TOPS 674
	ADM	8,1 SET FLAG - OV ON	TOPS 675
PTACC1	SRM	11,1, SAVE R,ES	TOPS 676
	TNOR	PTACC2	TOPS 677
	ADM	9,1 SET FLAG-OV ON AFTER R,ES	TOPS 678
			TOPS 679

Date:	11/14/64
Section:	8.4-J6-TOPS
Page:	37 of 39
Change:	2

PTACC2	SAM	11,1,	SAVE A-MOST	TOPS 680
	SAL	11,1,	SAVE A-LEAST	TOPS 681
	SEX	15	SAVE EXPONENT	TOPS 682
	TU	PTACC3		TOPS 683
	ADM	10,1	SET FLAG-Z ON	TOPS 684
PTACC3	CAD	F1		TOPS 685
	SAM	M11	SAVE F1	TOPS 686
	SFR	7,PTEMP5	SAVE EXPONENT	TOPS 687
	CAM	1,PTAMS1	LOAD FIRST MESSAGE	TOPS 688
	CALL	PTMSS	OUTPUT FIRST MESSAGE	TOPS 689
	LFR	5,PTEMP5+7		TOPS 690
	CAM	1,M15		TOPS 691
	CALL	PTSW5	OUTPUT A-MOST	TOPS 692
	CAM	1,PTAMS2	LOAD SECOND MESSAGE	TOPS 693
	JZM	8,PTACC4	'OV ON' FLAG NOT SET	TOPS 694
	SBM	1,5	ADJUST SECOND MESSAGE	TOPS 695
	CAD	13.	RESET OV	TOPS 696
	DIV	15,3,		TOPS 697
PTACC4	CALL	PTMSS	OUTPUT SECOND MESSAGE	TOPS 698
	LFR	5,PTEMP5+8		TOPS 699
	CAM	1,M15		TOPS 700
	CALL	PTSW5	OUTPUT A-LEAST	TOPS 701
	CAM	1,PTAMS3	LOAD THIRD MESSAGE	TOPS 702
	JZM	10,PTACC5	'Z ON' FLAG NOT SET	TOPS 703
	SBM	1,5	ADJUST THIRD MESSAGE	TOPS 704
PTACC5	CALL	PTMSS	OUTPUT THIRD MESSAGE	TOPS 705
	CAD	M11-3	FETCH R,ES	TOPS 706
	SAM	F5		TOPS 707
	SEX	15		TOPS 708
	CAM	0,PTAMS4	LOAD FOURTH MESSAGE	TOPS 709
	JUM	9,PTACC7	'OV ON AFTER R,ES' FLAG SET	TOPS 710
	JUM	10,PTACC6	'Z ON' FLAG SET	TOPS 711
	TU	PTACC8		TOPS 712

Date:	11/14/64
Section:	8,4-J6-TOPS
Page:	38 of 39
Change:	2

PTACC6 CAM 1,M15
 CALL PTSWS
 CAM 1,72
 CALL PTA
 TRA PTACC9
 PTACC7 ADM 15,128
 SBM 0,6
 PTACC8 CAM 1,M15
 CALL PTSWS
 PTACC9 CAM 1,M0
 CALL PTMSS
 CAM 1,PTEMP5+1
 CALL PTSW
 CAM 1,PTAMSS
 CALL PTMSS
 CAM 1,PTEMP5+9
 CALL PTSW
 CAD PTEMP5+1
 SAM PTEMP5+1
 CAD PTEMP5+7
 LAL PTEMP5+8
 LFR 7,PTEMP5
 CAE M15
 JZM 10,PTAC10
 ADE -128
 ADE -128
 PTAC10 LFR 7,PTEMP5+9
 OCTQ 10737
 LFR 5,PTEMP5+3
 LFR 6,PTEMP5+4
 LFR 7,PTEMP5+5
 LFR 4,PTEMP5+2
 JLH M3
 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

TOPS 713
 TOPS 714
 TOPS 715
 TOPS 716
 TOPS 717
 TOPS 718
 TOPS 719
 TOPS 720
 TOPS 721
 TOPS 722
 TOPS 723
 TOPS 724
 TOPS 725
 TOPS 726
 TOPS 727
 TOPS 728
 TOPS 729
 TOPS 730
 TOPS 731
 TOPS 732
 TOPS 733
 TOPS 734
 TOPS 735
 TOPS 736
 TOPS 737
 TOPS 738
 TOPS 739
 TOPS 740
 TOPS 741
 TOPS 742
 TOPS 743
 TOPS 744
 TOPS 745
 TOPS 746

Date: 11/14/64
 Section: 8.4-J6-TOPS
 Page: 39 of 39
 Change:

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
KO-IØLIST-OO-UI-AL

NAME: I/Ø 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 CØMMØN.

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

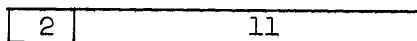
Format of the I/O List Word

Programmed by: C. W. Gear
Approved by:

CW Gear

Date: 7/20/64
Section: 8.4-KO-IØLIST
Page: 1 of 10
Change:

USE (Continued): The first quarter C of the list is the control word. It is split into 2 and 11 bit parts.



Control Quarter

The 11-bit group indicates the function to be performed:

0	Read
1000 ₈	Write
2040	Rewind
2050	Backspace
2120	End of File

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.

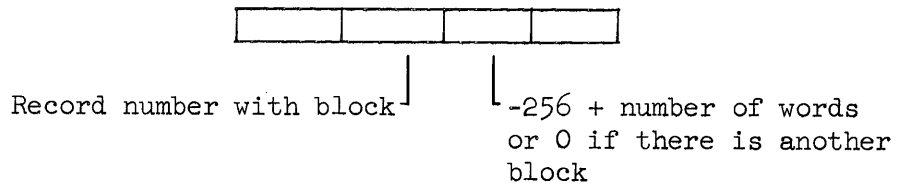
When the top bit of a control quarter is a zero, the list terminates. The CALL on the IØLIST program is said to be a partial or final CALL accordingly as the second bit of the control quarter is a one or a zero.

A partial CALL means that the next CALL on the IØLIST 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.

Date:	7/20/64
Section:	8.4-KO-IØLIST
Page:	2 of 10
Change:	

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:



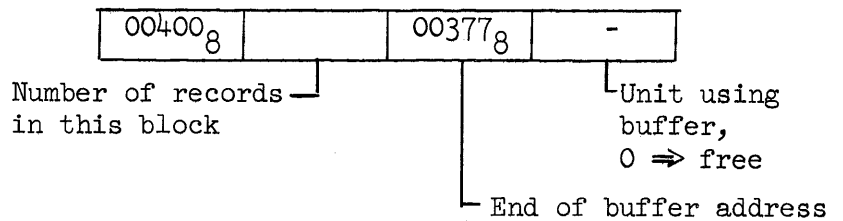
The words are placed into a buffer backwards until it is full (255 words) and then another record is started. When the last word of a block has been placed in the buffer, the control word is placed in the next available position and a record of $n + 1$ words is written where n is the number of data words 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.

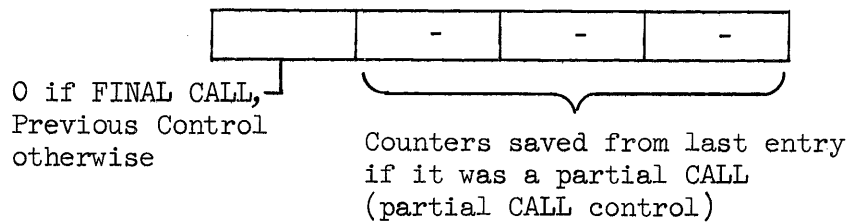
Date:	7/20/64
Section:	8.4-KO-IØLIST
Page:	3 of 10
Change:	

METHOD (Continued): A control word is used to tell whether the buffer is free or not, and if not, which unit is using it.



BUF+1: BUFFER CONTROL WORD

A control word is also used to remember whether the last CALL was partial or final.



BUF

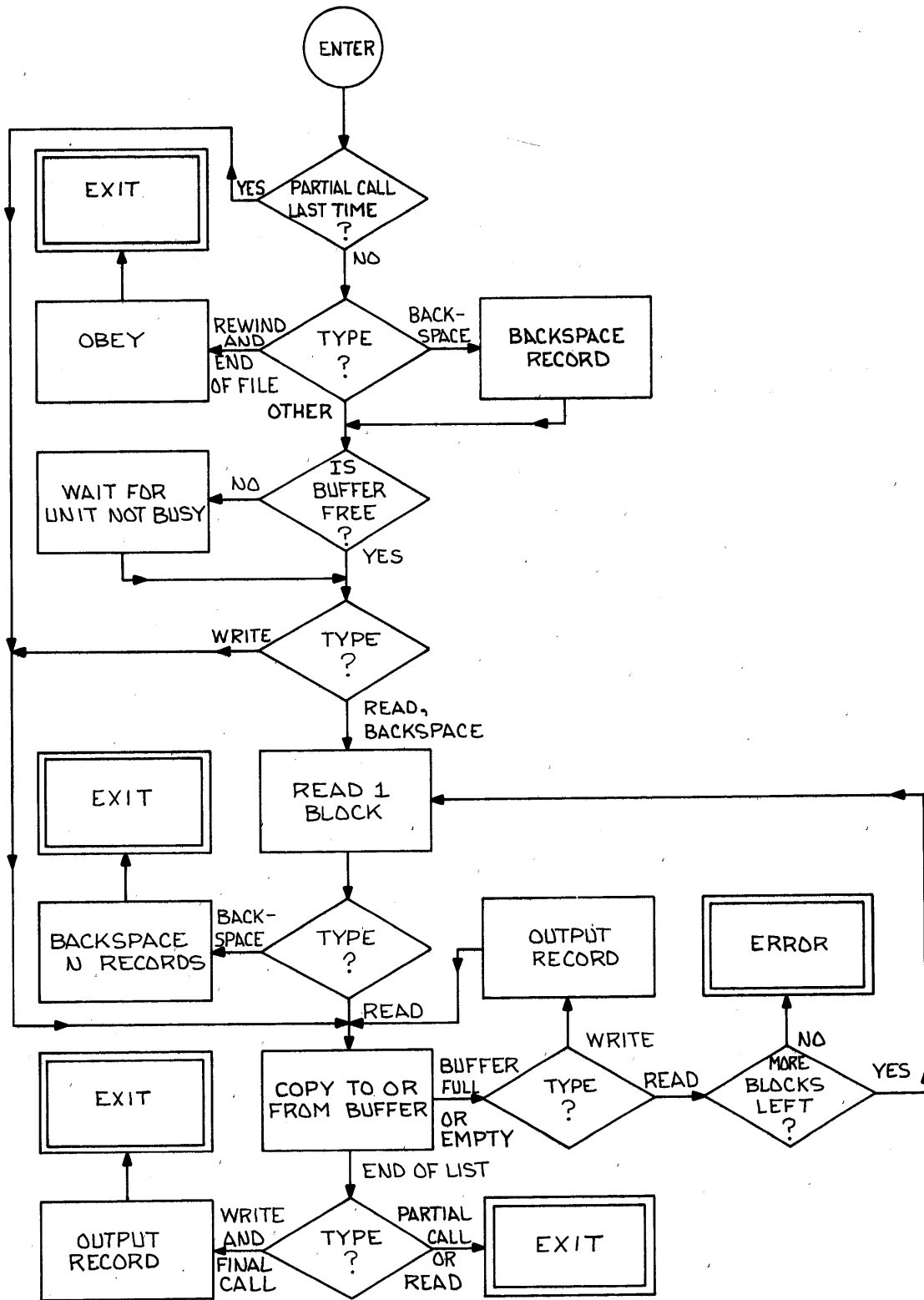
Use of Modifiers.

- 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.

Date: 7/20/64
 Section: 8.4-KO-IØLIST
 Page: 4 of 10
 Change:

METHOD (Continued): After a partial call, the information in F6 is stored in
BUF and restored at the next CALL.

Date:	7/20/64
Section:	8.4-KO-IØLIST
Page:	5 of 10
Change:	



IO LIST
SUBROUTINE FLOW

Date:	7/20/64
Section:	8.4-KO-IO LIST
Page:	6 of 10
Change:	

	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	SYS AUX
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

Date: 7/20/64
Section: 8.4-KO-IOLIST
Page: 7 of 10
Change:

IOL4	ANN	8,512	
	CAM	0	MO=0 MEANS INPUT
	CAM	10,-1	BLOCK COUNT EMPTY FOR INPUT
	JZM	0, IOL6A	INPUT
IOL13	CAM	10,-256	BLOCK EMPTY FOR OUTPUT
IOL6A	LFR	7, BUF+1	
	JZM	15, IOL3+1	BUFFER FREE
	SFR	7, IOL3	
	CAM	15	
	CALL	SYSAUX	
IOL3	BSS	1	WAIT FOR UNIT NOT BUSY
	ADM	13, 1	INCREASE RECORD COUNT
	SFR	7, BUF+1	
	CAM	9, M14	
IOL6	ADM	6,-1	WORD COUNT DECREASED
	JNM	6, IOL8	TEST FOR EMPTY
	CJZ	10, IOL10A	BUFFER EXHAUSTED
	JZM	0, IOL7	TEST INPUT
	CAD	5, 1,	
	SAM	M9	MOVE TO BUFFER
IOL5	SBM	9, 1	BUFFER COUNT
	TRA	IOL6	
	FIL		
IOL7	CAD	M9	MOVE FROM BUFFER
	SAM	5, 1,	
	TRA	IOL5	
IOL8	JPM	4, IOL9	JUMP IF LAST CONTROL
	ATN	1, 1,	NEXT CONTROL TO F5
	LFR	5	
	TRA	IOL6	
IOL9	CRN	4, 12	
	CAM	12	
	JPM	12, IOL10B	JUMP IF FINAL CONTROL
	ORM	8, 2048	
IOL12	SFR	6, BUF	RESTORE BUF

Date:	7/20/64
Section:	8.4-KO-IOLIST
Page:	8 of 10
Change:	

	LFR	4,T1	AND F4-F7
	LFR	5,T2	
	LFR	6,T3	
	LFR	7,T4	
	JLH	M3	
IOL 10B	JUM	0,IOL10	JUMP IF OUTPUT
	JPM	8,IOL20	NO BLOCKS LEFT JUMP
IOL 14	JPM	8,IOLERR	NO BLOCKS LEFT - ERROR
	LFR	7,IOL15	
	CAM	15,M11+1024	
	SFR	7,IOL15	
	CALL	SYSAUX	
IOL 15	DECQ	READ,RDBF-255,IOLEOF,0	READ TAPE
	CAM	12,WAIT	
	SFR	7,IOL16	
	CALL	SYSAUX	
IOL 16	BSS	1	WAIT UNTIL IN
	CRN	4,11	
	CAM	12	
	JNM	12,IOL21	BACKSPACE
	CAM	9,RDBF-255	
	LFR	7,M9	
	ORM	8,4096	
	JZM	14,IOL17	MORE BLOCKS TO COME
	ANM	8,4095	
	ADM	14,1	
IOL 17	ADM	6,1	RESTORE WORD COUNT
	ADM	9,M14+255	END OF BUFFER
	CAM	10,-256-M14	SET COUNT
	TRA	IOL6	
IOL 10A	JZM	0,IOL14	JUMP IF INPUT
IOL 10	LFR	7,BUF+1	
	CAM	14,M10	BUFFER COUNT
	SFR	7,M9	STORE CONTROL
	CAM	13,M9	

Date:	7/20/64
Section:	8.4-KO-IOLIST
Page:	9 of 10
Change:	

	CAM	12,WRITE	
	CAM	15,M11+1024	TAPE CODE
	SFR	7,IOL11	
	CALL	SYSAUX	
IOL11	BSS	1	WRITE ON TAPE
	LFR	7,BUF+1	
	CAM	15,M11+1024	SET BUSY BUFFER
	SFR	7,BUF+1	
	ADM	6,1	
	JZM	10,IOL13	
IOL20	CAM	8	SET M8=0 MEANING FINAL ENTRY
	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	DECQ	BCKSPR,0,0,0	
	CJU	11,IOL23	
	TRA	IOL20	
BUF	BSS	1	
	DECQ	WAIT,0,RDBF,0	
	ASSIGN	T1,T2,T3,T4	
RDBF	EQU	255	
IOLERR	CALL	SYSIO	
	DECQ	WRITE+2,MESS,0,0	
	CALL	SYSERR	
MESS	CHR	32, 6READ TAPE LIST TOO LONG.	

Date:	7/20/64
Section:	8.4-KO-IOLIST
Page:	10 of 10
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS
ILLIAC II LIBRARY PROGRAM
MO-CMPL-OO-UI-AL

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

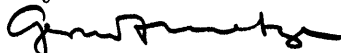
where PARAM is the address of a full word in memory containing the following parameters:

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

$$1 \leq N \leq 8192$$

Programmed by: W. J. Bouknight
Approved by:



Date: 9/22/64
Section: 8.4-MO-CMPL
Page: 1 of 3
Change:

USE (CONT'D):

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, M1 and M2 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 M1 = 301 and M2 = 501.

Date:	9/22/64
Section:	8.4-MO-CMP1
Page:	2 of 3
Change:	

	ENTRY	CMP1			CMP1 000
CMP1	SFR	4,COMMON	SAVE F4		CMP1 001
	SFR	6,COMMON+1	SAVE F6		CMP1 002
	SFR	7,COMMON+2	SAVE F7		CMP1 003
	LFR	4,M1	LOAD PARAMETERS		CMP1 004
	CSM	0,M0	SET COUNT FOR N WORDS		CMP1 005
CMP1A	ATN	1,1,	LOAD FIRST WORD		CMP1 006
	LFR	6	TO COMPARE		CMP1 007
	ATN	2,1,	LOAD SECOND WORD		CMP1 008
	LFR	7	TO COMPARE		CMP1 009
	EOM	8,M12	EOM M12 TO M8		CMP1 010
	EON	9,M13	EOM M13 TO M9		CMP1 011
	ORM	8	ORM M9 TO M8		CMP1 012
	EON	10,M14	EOM M14 TO M10		CMP1 013
	ORM	8	ORM M10 TO M8		CMP1 014
	EON	11,M15	EOM M15 TO M11		CMP1 015
	ORM	8	ORM M11 TO M8		CMP1 016
	JUM	8,CMP1B	JUMP IF NO COMPARE EQUAL		CMP1 017
	CJU	0,CMP1A	RETURN FOR MORE COMPARES		CMP1 018
	LFR	4,COMMON	RESTORE F4		CMP1 019
CMP1B	LFR	6,COMMON+1	RESTORE F6		CMP1 020
	LFR	7,COMMON+2	RESTORE F7		CMP1 021
	JLH	M3	RETURN		CMP1 022
	FIL				CMP1 023
COMMON	BSS	3	TEMPORARY STORAGE		CMP1 024

Date:	9/22/64
Section:	8.4-MO-CMP1
Page:	3 of 3
Change:	

DIGITAL COMPUTER LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

ILLIAC II LIBRARY PROGRAM

M2-PRINT-OO-UI-AL

M2-READ-OO-UI-AL

M2-PUNCH-OO-UI-AL

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)

Programmed by: M. Gaer
Approved by:

M. W. Gaer

Date: 7/16/64
Section: 8.4-M2-PRINT
Page: 1 of 62
Change:

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 CRN15, bit position + 1, putting the result into ML4 and checking the sign of ML4. $ML4 < 0 \Rightarrow$ 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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	2 of 62
Change:	

DESCRIPTION (Continued): / → BLSH: FRESET is called to finish this line and the next format character is read.

, → comma: This will have been encountered while setting up previous format instruction and will have terminated it. Next character is now read.

l - 10 → 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 MO.

(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 → 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	3 of 62
Change:	

DESCRIPTION (Continued): the S format is saved. If this is an output type blanks are output from M8, FLDBF is used; otherwise FRDBF is used to read in blanks.

X → BX: 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.

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 BSUB; 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.

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 FLDBF. 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, FRCNT, 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	4 of 62
Change:	

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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	5 of 62
Change:	

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 FRDBF, 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 F7, which is then

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	6 of 62
Change:	

DESCRIPTION (Continued):

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 FRDBF 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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	7 of 62
Change:	

DESCRIPTION (Continued): + → BPLUS (- enters inside 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.

FØRMRD

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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	8 of 62
Change:	

DESCRIPTION (Continued): is kept in FRCNT. The character is extracted, put into M8, 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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	9 of 62
Change:	

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 M8. Counts are then incremented and FCNTSC is saved.

FRESET

FRESET 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 INPBF. 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.

READDEC

READDEC 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. Normalization 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 normalization 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 0000-1111 corresponding to 10^0-10^{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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	12 of 62
Change:	

DESCRIPTION (Continued): The following is a description of the modifiers and the various sections of the program.

Modifier Use

- M0 GACVT puts converted digit -9 into M0.
GAØUT1 moves digit to M4.
- M1 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 GAØUT1.
- 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 M1 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. GAØUT 1 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 GAØUT1 is entered, a sign is printed.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	13 of 62
Change:	

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 GA~~O~~UTL, 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 GA~~O~~UTL 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).

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	14 of 62
Change:	

DESCRIPTION (Continued):

- M11 Contains -1 minus the number of digits, zeros and blanks which precede the point. GALD and GAØUT1 automatically insert the point when this count reaches 0.
- M12 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.
- M13 Holds the exponent of the printed number.
- M14 Is not used.
- M15 Negative for E fields and M15 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 $\frac{1}{2} 10^{-n}$ 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 M11. The lead

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	15 of 62
Change:	

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 ERRØRT is made.

Subroutines in FDP

- GALD prints the lead zeros and blanks and inserts the point and sign if necessary.
- GAØUT1 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.

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	16 of 62
Change:	

F4:	M0	M1	M2	M3
	Format Character Count	I/O List Word Address	Type: Input/Output Used for Field Length in E, F, I	Return

F5:	M4	M5	M6	M7
	0 Repeat or 1 Reentry Bit (0 = + ⇒ Last 1 = - ⇒ More)	Address in Core of Data Block	Number of Words	First Format Address

F5 filled from I/O List Word.

F6:	M8	M9	M10	M11
	Format Character Symbol Read	Temporary Count (Decimal Place Count for E, F Also)	Multiplicity of Current Format Instruction, e.g., 5 of 5E15,6	Repetition Count for Parenthesis

F7:	M12	M13	M14	M15
	Scaling Power	Address of Push Down Bits of Parenthesis	For Interrogating Switches in M15	Switches in Low Bit Position. Also + ⇒ F, - ⇒ E

1 ⇒ on, 0 ⇒ off.

Figure 1. Modifier Layout for Master Control Logic

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	17 of 62
Change:	

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

<u>Representation--Actual Character</u>	<u>Representation--Actual Character</u>
0 = blank	39 = P
1-9 = 1-9	40 = Q
10 = 0	44 = *
17 = /	48 = +
18 = S	49 = A
23 = X	52 = D
27 = ,	53 = E
28 = (54 = F
32 = -	56 = H
35 = L	57 = I
36 = M	60 =)

Figure 3. BCD Characters

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	18 of 62
Change:	

F4:

M0	M1	M2	M3
Left, Right Switch for Correct 1/2 of 1/4 Word	Format Character Count	1/4-Word Count in Format Word	Current Format Word Address

Figure 4. FRCNT

F5:

M4	M5	M6	M7
Total Buffer Character Count -133 or -80	Address in Buffer Where FBFWD Is to Be Placed	Left or Right 1/2 or 1/4 Word -1 ⇒ Left, 0 ⇒ Right	Quarter Word Count

Figure 5. FCNTS

F5:

M4	M5	M6	M7
Total Character Count -80	Address of Current Word in Input Buffer	Right or Left 1/2 of Quarter Word. -1 ⇒ Left, 0 ⇒ Right	Quarter Word Count

Figure 6. FCNTSC

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	19 of 62
Change:	

F4:

M0	M1	M2	M3
Not Used	Number of Places Past Decimal Point	Length of Field	Return Address

F5:

M4	M5	M6	M7
Exponent in Here	Last Character Read	Original Return Saved When Using FRDBF or FEXP	Length of Field

F6:

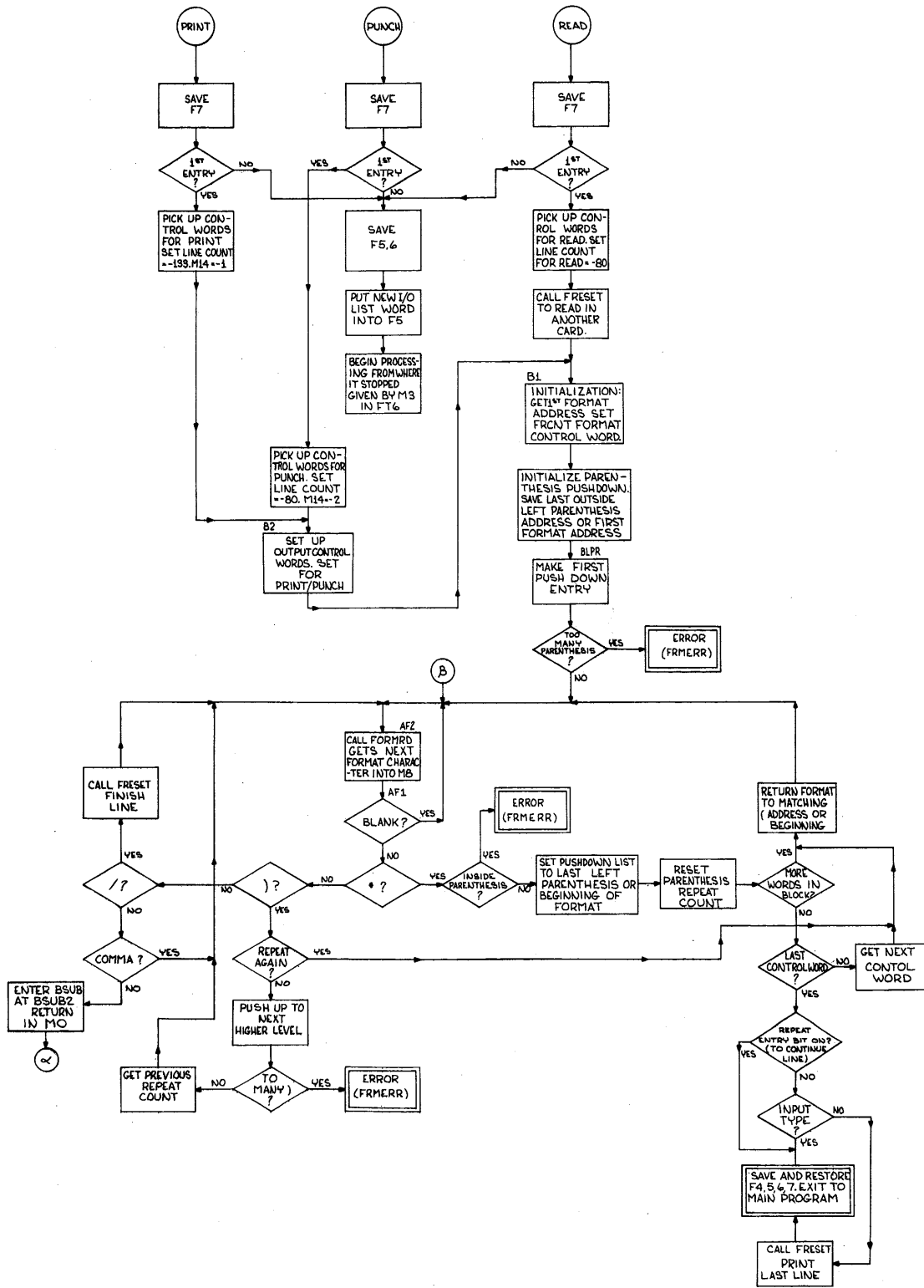
M8	M9	M10	M11
Character Read	= 1 ⇒ Next Sign for Exponent = 0 Initially	0 ⇒ Exponent Plus -1 ⇒ Exponent Minus	+1 ⇒ Number + -1 ⇒ Number -

F7:

M12	M13	M14	M15
= 1 ⇒ in Exponent = 0 Initially	= 1 ⇒ Digit Read is First Digit of Number	= 1 ⇒ Digit Read is the First Digit of the Exponent	= 1 ⇒ Decimal Point Has Been Encountered

Figure 7. READEC Modifiers

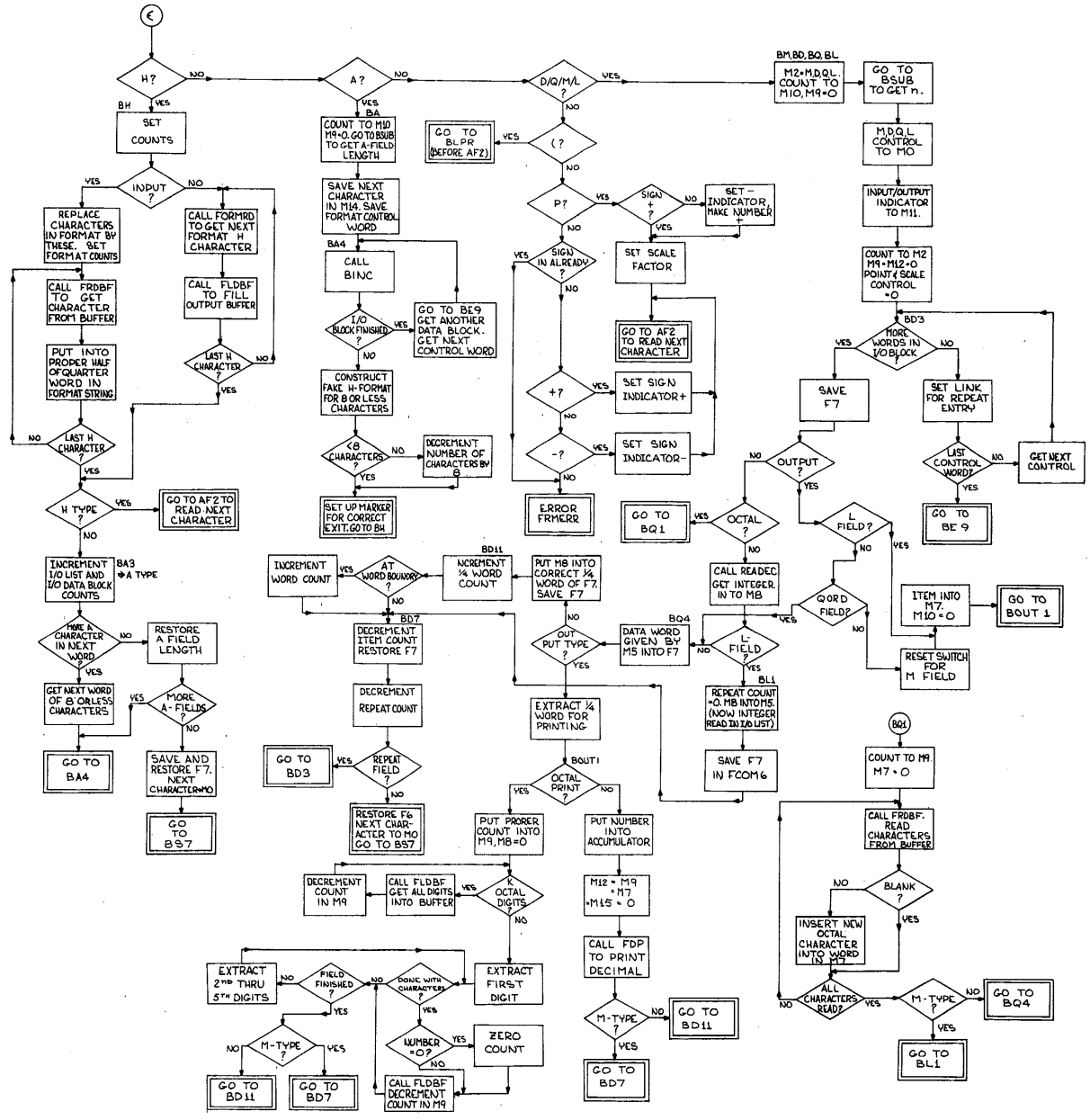
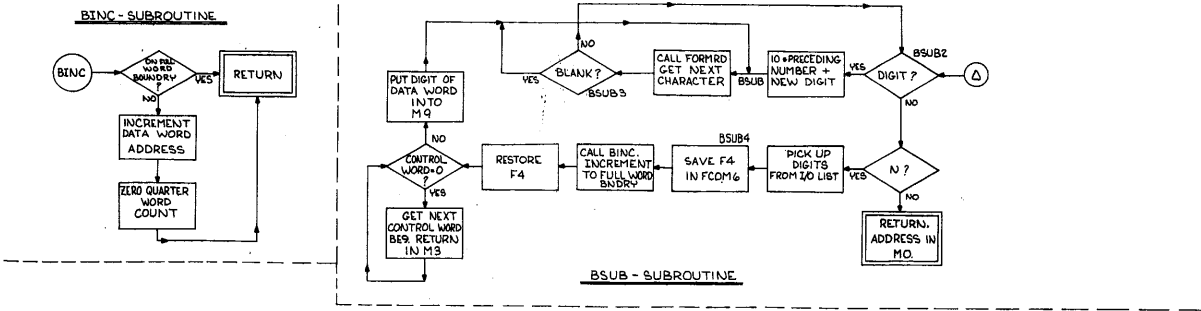
Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	20 of 62
Change:	



MAIN FORMAT SCAN CONTROL

PART 1

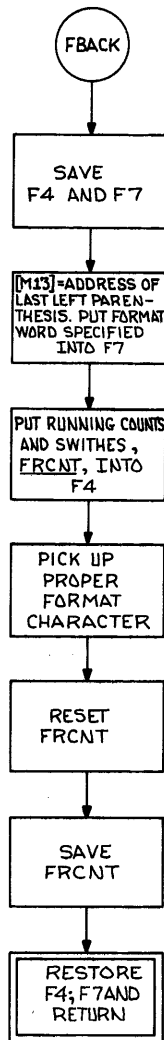
Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 21 of 62
 Change:



MAIN FORMAT SCAN CONTROL

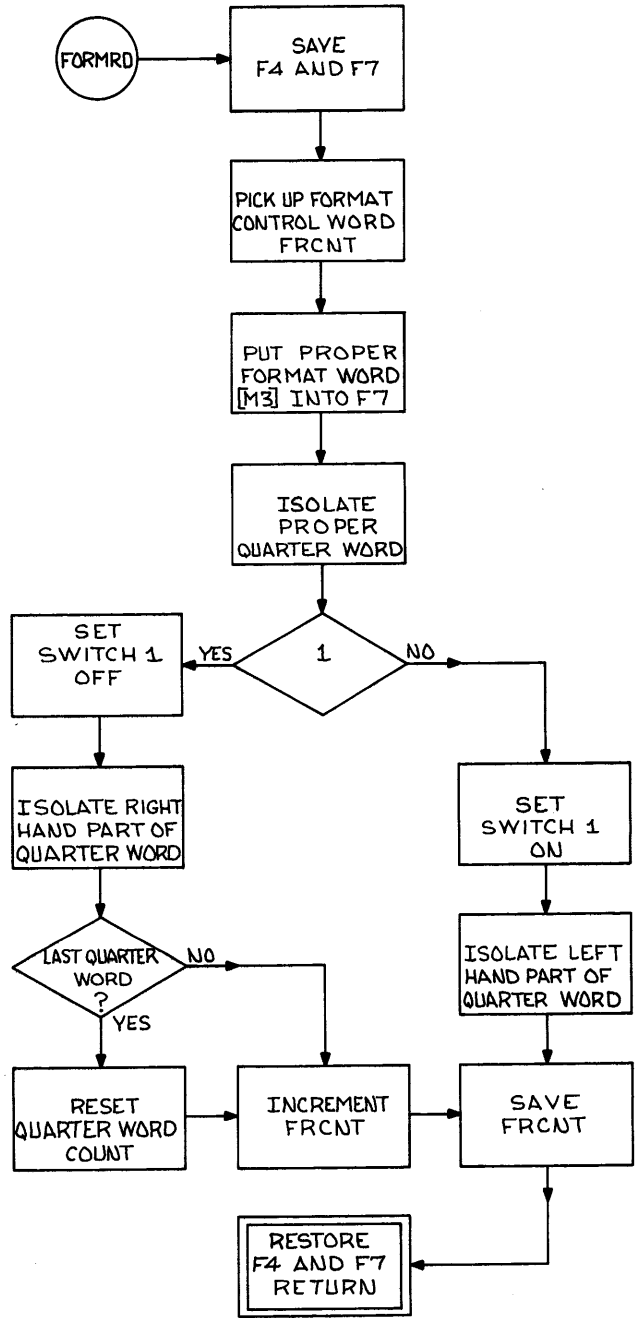
PART 3

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 23 of 62
 Change:



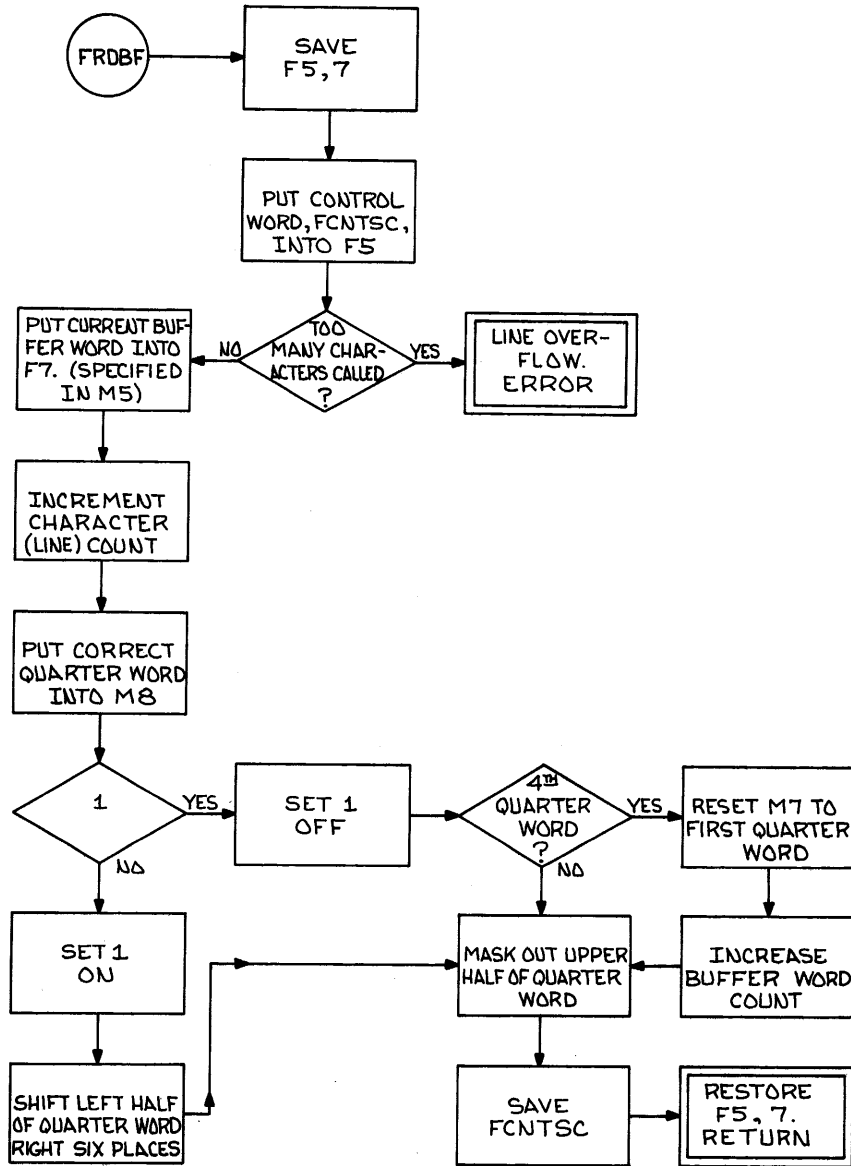
FBACK

Date: 7/16/64
Section: 8.4-M2-PRINT
Page: 24 of 62
Change:



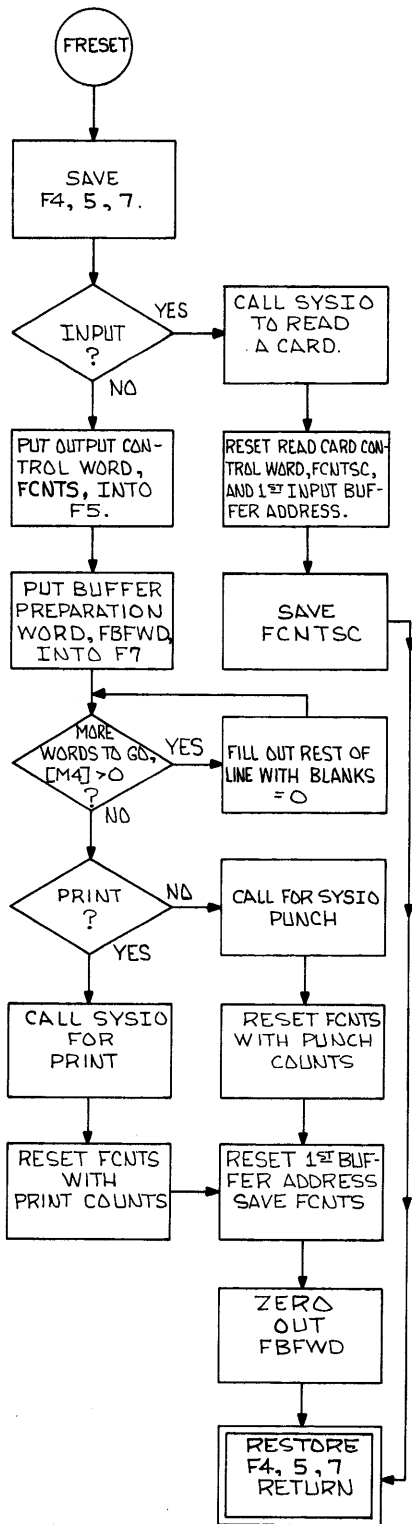
FORMRD

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 25 of 62
 Change:



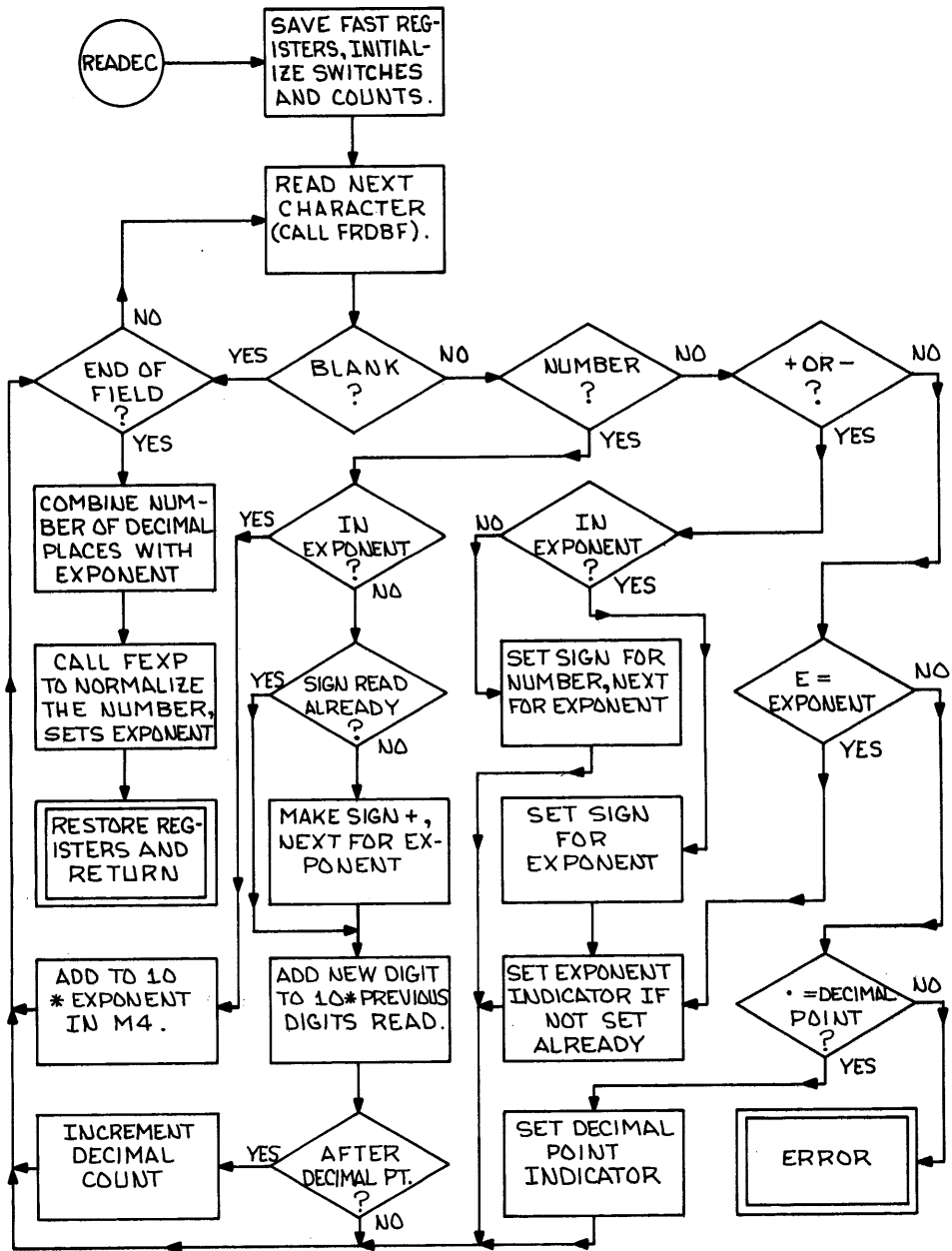
FRDBF

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 27 of 62
 Change:



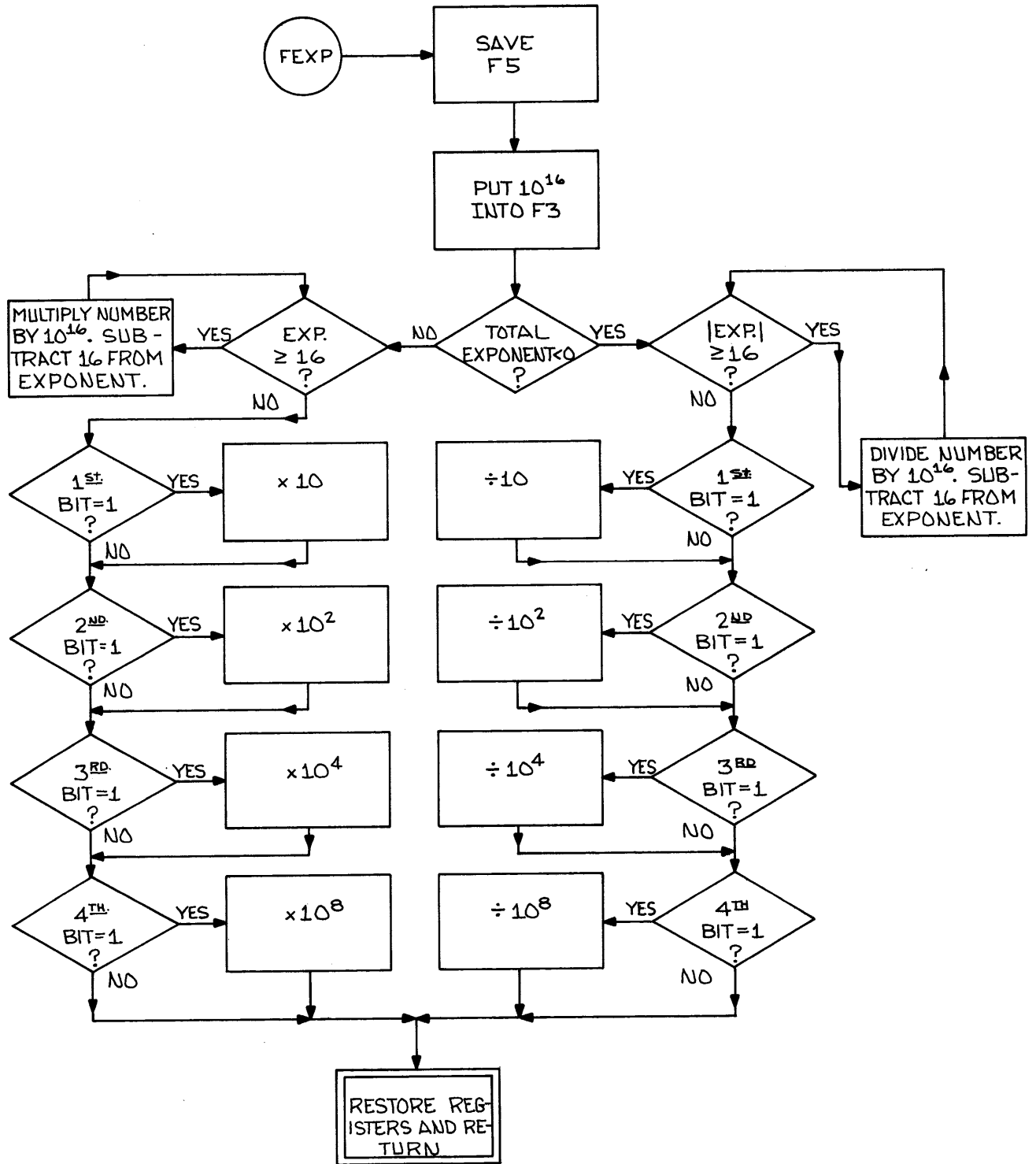
FRESET

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 28 of 62
 Change:



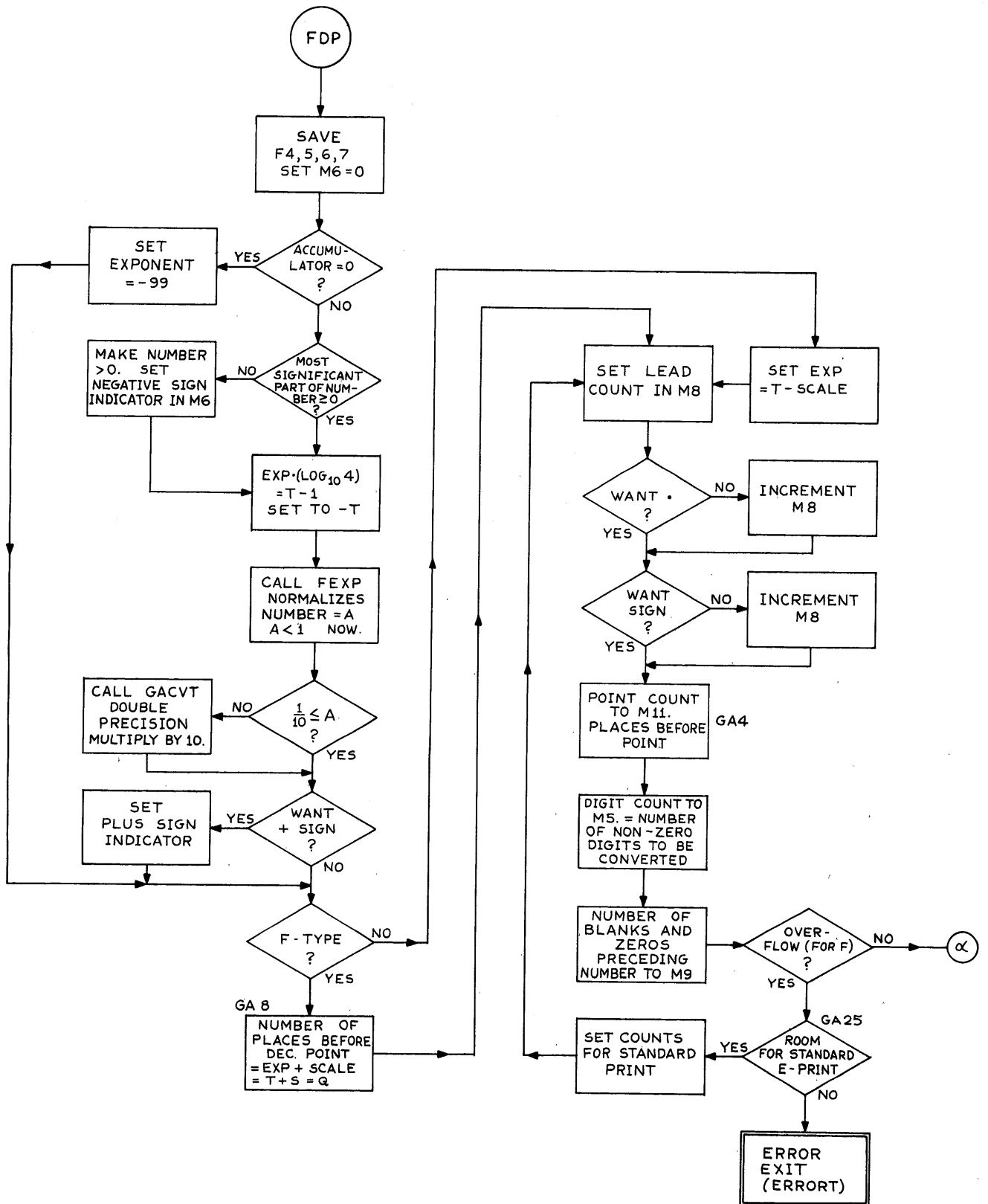
READEC

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 29 of 62
 Change:

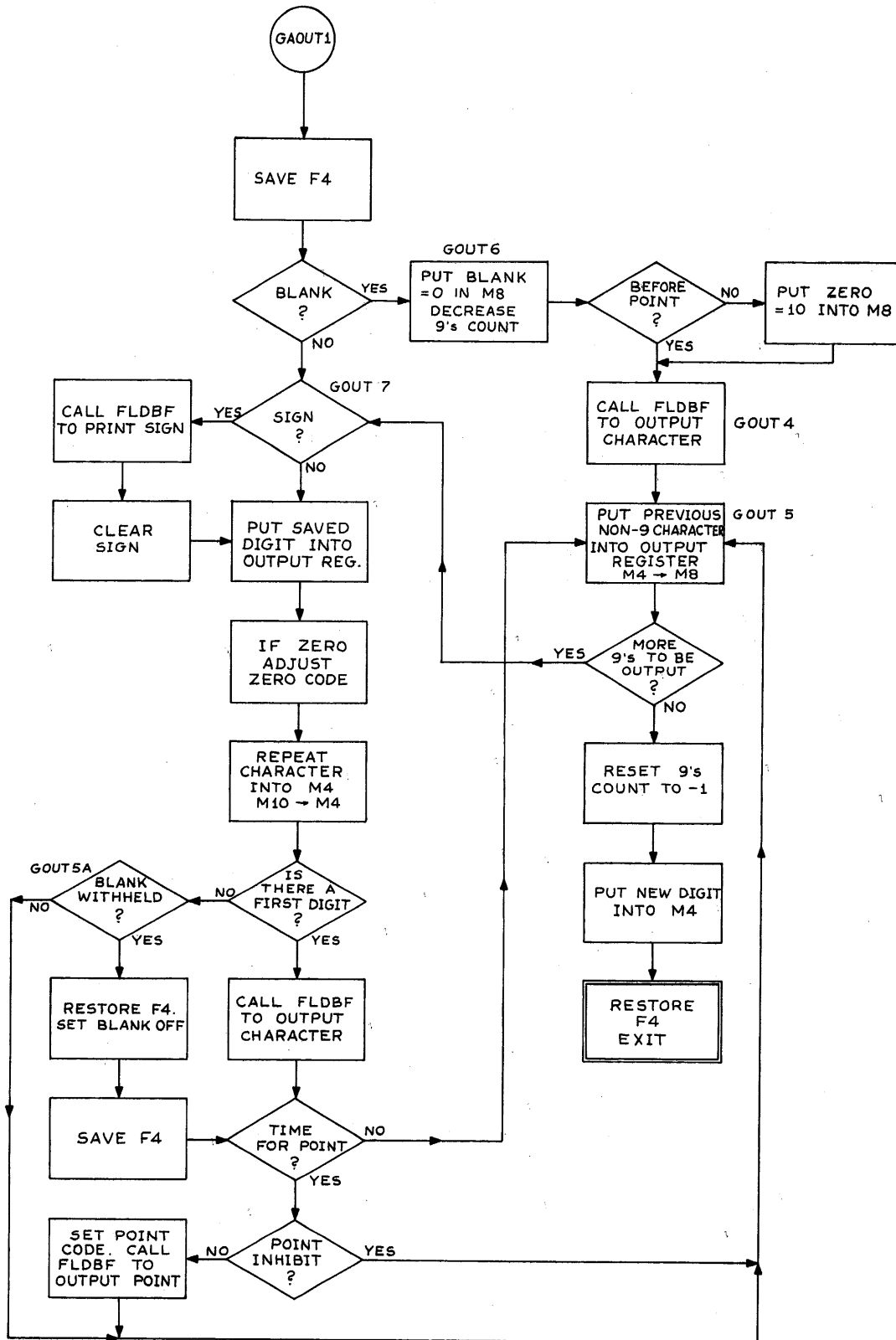


FEXP

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 30 of 62
 Change:



Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 31 of 62
 Change:



Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 33 of 62
 Change:

	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	
	LFR	7, CON	M12=M15=0, M113=M14=1
	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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	34 of 62
Change:	

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	
J6	SBM	4,M12+M1	NORMALIZE WITH PROPER EXPONENT
	CALL	FEXP	
END	LFR	4,FCOM29	
	JPM	11,U	PLUS
	STC	2,3,	MAKE NUMBER MINUS
	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,EXPRT	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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	35 of 62
Change:	

	XCH	2,3,	
	MPY	10.	
	ADD	2,3,	PLUS NEW DIGIT
	ADD	10+M8.	
	JZM	15,X	AFTER DEC. PT.
	ADM	1,1	INCREMENT DECIMAL COUNT
	TRA	X	
EXPR	JZM	14,H	IF NO SIGN FOR EXPONENT, SET PLUS
	CAM	10,1	
	CAM	14	SET EXPONENT SIGN IN
H	CRM	4,10	10 X PREVIOUS EXPONENT DIGITS
	CRM	4,3,2	
	ADM	4,10+M8	ADD NEW DIGIT
	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	SAVE FAST REGISTERS
	SFR	5,GAT+1	
	SFR	6,GAT+2	
	SFR	7,GAT+3	
	CAM	6	
	TZ	GA22A	ZERO
	STC	F2	
	XCH	F2	STORE NUMBER
	TZP	GA1	POSITIVE
	STN	F0	CHANGE SIGN
	SSC	F2	
	CAM	6,4096	SET NEGATIVE INDICATOR
	XCH	F2	
GAI	STR	F3	
	SEX	3	
	CAD	M3.	
	MPY	GAT+4	EXPONENT * LOG BASE 10 OF 4

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	36 of 62
Change:	

	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	
GA2	CAD	F3	
GA23	CAM	1	SET NO SPARE BLANK
	ANN	15,16	
	ORM	6	SET PLUS SIGN OR IGNORE
	JPM	15,GA8	F TYPE
	CAM	13,M8+1-M12	EXP= T-S
GA26	CAM	8,3	
GA9	ADM	8,M9-M2+M12	LEAD COUNT
	JPM	7,GA3	NO POINT
	ADM	8,1	
GA3	JZM	6,GA4	NO SIGN
	ADM	8,1	
GA4	CAM	11,M8-M12	POINT COUNT
	CAM	5,M12+M9	DIGIT COUNT
	JPM	5,GA5	POSITIVE
	SBM	8,M5	DECREMENT LEAD COUNT
GA5	CSM	5,M5	
	CAM	9,M8	-1-LEAD COUNT TO M9

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	37 of 62
Change:	

	JPM	9,GA25	LEAD COUNT NEGATIVE
	JPM	11,GA25	POINT TOO FAR TO LEFT
	JNM	15,GA6	E FIELD
	CJZ	9,GA7	SAVE BLANK FOR F FIELD
	CAM	1,4096	SET SAVED
GA6	CAM	10	LEAD CHARACTER BLANK
	CAM	4,4096	NO DIGIT YET
GALD	CJU	11,GALD3	NOT TIME FOR POINT
	JZM	6,GALD2	NO SIGN
	CAM	8,32	MINUS
	JNM	6,GALD1	CORRECT
	CAM	8,48	PLUS
GALD1	CALL	FLDBF	OUTPUT SIGN
	CAM	6	CLEAR SIGN INDICATOR
GALD2	JPM	7,GALD5	NO POINT
	CAM	8,59	
	CALL	FLDBF	OUTPUT POINT
GALD5	CJZ	9,GALD6	DONE ON LEAD CHARS
	CAM	10,10	CHARACTER NOW ZERO
	JNM	15,GALD4	E FIELD
	CAM	4	FIRST DIGIT ZERO
	JPM	1,GALD3	NO SPARE
	CAM	1	CLEAR SPARE INDICATOR
	TRA	GALD4	
GALD3	CJZ	9,GALD6	DONE ON LEAD CHARACTERS
GALD4	CAM	8,M10	
	CALL	FLDBF	OUTPUT LEAD CHARACTER
	TRA	GALD	LOOP
GA7	CAM	9,-1	NO SPARE IN F FIELD
	TRA	GA6	
GA8	ADM	12,M8+1	Q=S+T
	CAM	8,-1	
	TRA	GA9	
GALD6	CAM	9,-1	
	CAM	10,9	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	38 of 62
Change:	

	JPM	5,GA27		NO DIGITS OUT
GA10	CALL	GACVT		CONVERT FIRST DIGIT
	CJU	5,GA10		NINE, SAVE
	TRA	GA18		DONE WITH DIGITS
	CALL	GAOUT1		OUTPUT SAVED DIGITS
	CJU	5,GA10		
	SUB	10,3,2048		ROUND LAST DIGIT...
	TN	GA19		NO ROUNDING NECESSARY
	ADM	4,1		ROUND
GA19	CALL	GAOUT1		OUTPUT LAST DIGIT
	JPM	15,GA17	F TYPE	
GA28	CAM	8,53		
	CALL	FLDBF		OUTPUT E,
	CAM	8,48		
	JPM	13,GA12	EXPONENT SIGN	
	CAM	8,32		
	CSM	13,M13		
GA12	CALL	FLDBF		
	CAM	8		
GA13	ADM	13,-10		
	JNM	13,GA14		
	CJU	8,GA13		
GA14	JUM	8,GA15		
	CAM	8,10		
GA15	CALL	FLDBF		AND EXPONENT DIGITS
	CAM	8,M13+10		
	JUM	8,GA16		
	CAM	8,10		
GA16	CALL	FLDBF		
GA17	LFR	4,GAT		RESTORE FAST REGISTERS
	LFR	5,GAT+1		
	LFR	6,GAT+2		
	LFR	7,GAT+3		
	JLH	M3		EXIT
GA18	SUB	10,3,2048		DO WE ROUND LAST 9

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	39 of 62
Change:	

	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
GA21	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
GADUT1	SFR	4,GAT1
	JNM	1,GOUT6
GOUT7	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.

EXIT
SAVE F4
BLANK SAVED
NO SIGN
MINUS

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	40 of 62
Change:	

	JNM	6,GOUT1	CORRECT
	CAM	8,48	PLUS
GOUT1	CALL	FLDBF	OUTPUT SIGN
	CAM	6	CLEAR SIGN
GOUT2	CAM	8,M4	SAVED DIGIT TO M8
	JUM	8,GOUT3	CODE CORRECT ZERO
	CAM	8,10	
GOUT3	CAM	4,M10	REPEAT CHARACTER TO M4
	JNM	8,GOUT5A	NO FIRST DIGIT
	CALL	FLDBF	OUTPUT M8
GOUT8	CJU	11,GOUT5	NOT TIME FOR POINT
	JPM	7,GOUT5	
	CAM	8,59	
GOUT4	CALL	FLDBF	OUTPUT POINT
GOUT5	CAM	8,M4	M4 TO OUTPUT REGISTER
	CJU	9,GOUT7	NOT DONE YET
	CAM	9,-1	RESET NINES COUNT
	CAM	4,M0+9	NEW DIGIT TO M4
	LFR	4,GAT1	RESTOR F4
	JLH	M3	EXIT
GOUT6	CAM	8	BLANK CLEAR SPARE BLANK INDICATOR
	ADM	9,-1	
	JNM	11,GOUT4	
	CAM	8,10	
	TRA	GOUT4	
GOUT5A	JPM	1,GOUT5	
	LFR	4,GAT1	
	CAM	1	
	SFR	4,GAT1	
	TRA	GOUT8	
GA22A	CAM	8,-100	EXPONENT OF ZERO IS -99
	STR	F2	
	TRA	GA23	
GA24	CAM	0,35	PRINT * BECAUSE NO SPACE
	ADM	9,1	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	41 of 62
Change:	

	CALL	GAOUT1	
	TRA	GA19	
GA25	CAM	9,M2-6	
	JNM	9,ERRORT	NO ROOM FOR STANDARD PRINT
	CAM	7,4096	SET UP STANDARD PRINT
	ORM	6,32	
	CAM	13,M12	OF S=0,E+N.(N-6),
	CAM	12	
	ORM	15,4096	
	TRA	GA26	
GA27	JPM	4,GA30	DIGIT ALLREADY CONVERTED
	JNM	15,GA28	E TYPE, PRINT EXPONENT
	JPM	1,GA17	NO SPARE
	CAM	1	CLEAR SPARE
	CAM	4	
GA30	SUB	10,3,2048	DIGIT IS ZERO
	TN	GA19	NO ROUNDING
	JUM	5,GA19	NO ROUNDING TOO SMALL
	ADM	4,1	ROUND
	TRA	GA19	
GAT	BSS	4	TEMPORARY STORAGE
	OCTQ	04642,00465,00522,14000	LOG 4 BASE 10
	OCTQ	03146,06314,14631,11577,03146,06314,14631,11551	1/10
FEXP	SFR	5,FEXPC1	SUBROUTINE TO NORMALIZE DECIMALS NUMBERS
	LFR	3,FTABL	10 SIXTEENTH INTO F3
	JNM	4,FEXP9	EXPONENT NEGATIVE
	STC	2,3,	DOUBLE PRECISION MULTIPLY BY
	TRA	FEXP1	10 SIXTEENTH
FEXP2	MPY	3,3,	
	XCH	2,3,	
	MPY	3,3,	
	ASC	2,3,	
FEXP1	SBM	4,16	EXPONENT GREATER THAN 16
	CSM	5,4	
	JPM	4,FEXP2	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	42 of 62
Change:	

FEXP3	CRM	4,1	POWERS OF 10 LESS THAN 16
	JPM	4,FEXP4	PRESENT UN EXPONENT
	MPY	5,FTABL	DOUBLE PRECISION MULTIPLY BY
	STC	0,3,	PROPER POWER OF 10
	CAD	1,3,	
	MPY	2,3,	
	ADD	0,3,	
	STC	2,3,	
FEXP4	CJU	5,FEXP3	GET NEXT HIGHER POWER OF 10
	ADD	2,3,	
	TRA	FEXP10	
FEXP9	SBM	4,1	COMPLEMENT EXPONENT
	TRA	FEXP5	
FEXP6	DIV	3,3,	DOUBLE PRECISION DIVIDE BY
	SRM	2,3,	10 SIXTEENTH
	XCH	2,3,	
	DIV	3,3,	
	ADD	2,3,	
FEXP5	ADM	4,16	EXPONENT LESS THAN -16
	JNM	4,FEXP6	
	CSM	5,4	
FEXP7	CRM	4,1	-POWER OF 10 GREATER THAN -16
	JNM	4,FEXP8	
	DIV	5,FTABL	DOUBLE PRECISION DIVIDE BY
	SRM	2,3,	PROPER POWER OF 10 LESS THAN 16
	STR	0,3,	
	CAD	1,3,	
	VID	2,3,	
	ADD	0,3,	
FEXP8	CJU	5,FEXP7	
FEXP10	LFR	5,FEXPC1	
	JLH	M3	
FEXPC1	BSS	1	
	OCTQ	05000,00000,00000,00002	10
	OCTQ	03100,00000,00000,00004	10 SQUARED

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	43 of 62
Change:	

	OCTQ	04704,00000,00000,00007	10	FOURTH
	OCTQ	02765,16040,00000,00016	10	EIGHTH
FTABL	OCTQ	04341,13623,07701,00033	10	SIXTEENTH
READ	SFR	7,FCOM4		
	LFR	7,FCOM1		
	JNM	12,RD1		REPEAT ENTRY
	SFR	4,FCOM1		
	CAM	15,512		SET FOR READ
	CALL	FRESET		READ FIRST CARD
	TRA	B1		
RD1	LFR	7,FCOM3		REPEAT ENTRY START
	SFR	6,FCOM3		
	LFR	6,FCOM2		
	SFR	5,FCOM2		
	ATN	1,1,		
	LFR	5		NEW I-O LIST WORD
	LDM	7,FCOM1		
	SFR	4,FCOM1		
	LFR	4,FT6		
	JLH	M3		
PUNCH	SFR	7,FCOM4		
	LFR	7,FCOM1		
	JNM	12,RD1		REPEAT ENTRY
	CAM	12,-80		COUNT FOR PUNCHED CHARACTERS
	CAM	13		
	CAM	14,-2		
	CAM	15,M		
	TRA	B2		
PRINT	SFR	7,FCOM4		
	LFR	7,FCOM1		
	JNM	12,RD1		REPEAT ENTRY
	CAM	12,-133		COUNT FOR PRINTED CHARACTERS
	CAM	13,256		
	CAM	14,-1		
	CAM	15,2+M		

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	44 of 62
Change:	

B2	SFR	4,FCOM1	
	CAM	3,M13	
	CAM	13,OUTBF	OUTPUT CONTROL WORD
	SFR	7,FCNTS	
	CAM	15,M3	SET PRINT OR PUNCH CONTROL
B1	SFR	6,FCOM3	
	LDM	11,M1	FORMAT START ADDRESS
	CAM	8,-2	
	CAM	9	
	CAM	10,-4	
	SFR	6,FRCNT	FORMAT CONTROL WORD SET
	CAM	12	SET SCALE TO ZERO
	SFR	6,PUSHDN+2	INITIALIZE PARENTHESIS PUSH DOWN
	CAM	13,PUSHDN	M13 IS PUSH DOWN ADDRESS
	SFR	5,FCOM2	
	ATN	1,1	
	LFM	5	FIRST CONTROL WORD
BLPR	CAM	10,M11	LEFT PARENTHESIS AND START OF FORMAT
	CAM	11,M9-1	MAKE FIRST PUSH DOWN ENTRY
	LDM	9,FRCNT	
	CAM	8,M12	
	SBM	13,PUSHDN+4	TOO MANY NESTED PARNETHESES
	JPM	13,FRMERR	
	ADM	13,PUSHDN+5	
	SFR	6,M13	STORE ENTRY
AF2	CAM	7	M7 IS UDED TO DETERMINE WHICH CHARS. MAY BE
AF3	CAM	9	READ. NUMBERS ARE CONSTRUCTED IN M9
AF5	CALL	FORMRD	READ FORMAT CHAR.
AF1	JZM	8,AF5	SKIP BLANK CHARACTER
	ADM	8,-44	
	JZM	8,BSTAR	ASTERISC
	ADM	8,-16	
	JZM	8,BRPR	RIGHT PARNETHESUS
	ADM	8,43	
	JZM	8,BSLSH	SLASH. END OF LINE

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 45 of 62
 Change:

ADM	8,-10	
JZM	8,AF2	COMMA
JNM	7,FRMERR	SHOULD HAVE BEEN TERMINATING CHAR.
ADM	8,16	
JSB	0,BSUB2	DIGIT, GO CONVERT
FIL		
ANN	7,2	
CAM	14	
JUM	14,AF4	MUST BE P SINCE + OR - OCCURRED
ADM	8,-7	
JZM	8,BS	S FOR SPACES
ADM	8,-5	
JZM	8,BX	X FOR SPACE
ADM	8,-34	
ANM	15,4077	SET INHIBIT PLUS AND F SWITCHES
JZM	8,BEF	F FIELD
ADM	8,3	
JZM	8,BEF	I WHICH IS SAME
ORM	15,4096	SET E SWITCH
CJZ	8,BEF	E FIELD
ADM	8,-3	
JZM	8,BH	HOLLERITH FIELD
ADM	8,7	
CAM	2	
JZM	8,BA	A FIELD
ADM	8,-3	
JZM	8,BD	D FIELD
CJZ	8,BA	
ADM	8,11	
JZM	8,BQ	Q FIELD FOR OCTAL QUARTER WORDS
ADM	8,4	
JZM	8,BM	M FIELD FOR OCTAL LOCATIONS
CJZ	8,BL	L FIELD FOR DECIMAL LOCATIONS
ADM	8,7	
JZM	8,BLPR	LEFT PARENTHESIS

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	46 of 62
Change:	

	ADM	8,17	
AF4	ADM	8,-28	
	JZM	8,BP	POWER OR SCALE FIELD
	JUM	14,FRMERR	ALREADY HAD SIGN
	ADM	8,-9	
	JZM	8,BPLUS	PLUS SIGN
	ADM	8,16	
	JUM	8,FRMERR	NOT MINUS SIGN
	ORM	7,4	SET MINUS
BPLUS	ORM	7,2	SET SIGNED
	TRA	AF5	
BINC	ANN	4,3	SUBROUTINE TO MOVE I-O LIST TO FULL WORD
	CAM	14	
	JZM	14,BINC1	ALREADY ON FULL WORD
	ADM	5,1	INCREMENT WORD ADDRESS
	ANM	4,8188	QUARTER WORD ADDRESS NOW ZERO
BINC1	JLH	3,0	EXIT
BRPR	SBM	11,1	RIGHT PARENTHESIS, COUNT REPEAT COUNT
	JNM	11,BRPR1	REPEAT AGAIN IF POSITIVE
	FIL		
BSTAR1	CALL	FBACK	FORMAT BACK UP TO MATCHING LEFT PARENTHESIS
	TRA	AF2	
BRPR1	SBM	13,PUSHDN+2	PUSH UP TO NEXT HIGHER LEVEL
	JNM	13,FRMERR	TOO MANY CLOSING PARNTHESES
	ADM	13,PUSHDN+1	
	LDM	10,M13+1	
	CAM	11,M10	FETCH PREVOIUS REPEAT COUNT
	TRA	AF2	
BSLSH	CALL	FRESET	SLASH, NEXT LINE OR CARD
	TRA	AF2	
BP	ANN	7,4	POWER.
	CAM	14	
	JZM	14,BP1	POSITIVE SIGN
	CSM	9,M9	CHANGE SIGN
BP1	CAM	12,M9	SET SCLAE FACTOR

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	47 of 62
Change:	

	TRA	AF2
BEF	CAM	10,M9-1
	CAM	2
	CAM	7
	CAM	9
BEF1	CALL	FORMRD
	CAM	14,M8-37
	CAM	0,BE2
	JZM	14,BSUB4
	ADM	14,26
	JNM	14,BSUB3
	ADM	14,-37
	EOM	15,2
	JUM	14,BEF1
	EOM	15,18
BE1	JSB	0,BSUB
	FIL	
BE2	JPM	7,BE3
BE7A	CAM	0,M8+11
	FIL	
BE7	CALL	BINC
	JZM	6,BE4
	ANN	15,512
	CAM	14
	JUM	14,BE5
	CAD	5,1,
	CRN	15,2
	CAM	14
	JPM	14,BEA1
	ADD	5,1,
BEA1	CALL	FDP
BE6	ADM	6,-1
	ADM	10,-1
	JPM	10,BE7
	CAM	12

E F OR I FIELD

READ NEXT CHAR

N CHARACTER, GO GET NUMBER FROM IO LIST

DIGIT, GO READ REST OF NUMBER

SET DOUBLE PRECISION BIT

DOUBLE PRECISION

CANCEL DOUBLE PRECISION AND SET PLUS PRINT

GO READ A NUMBER

POINT NET READ YET

SAVE NEXT FORMAT CHAECTAER

INCREMENT WORD COUNT IF NOT ON WORD BOUNDARY

IO WORD EXHAUSTED

READ BIT

BRANCH IF INPUT

NUMBER TO ACCUMULATOR

DOUBLE PRECISION BIT

BRANCH IF SINGLE PRECISION

SECONG HALF OF NUMBER

PRINT

DECREASE COUNT

DECREASE E FIELD COUNT

REPEAT IF NOTE NUMBERS

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	48 of 62
Change:	

BS7	CAM	7,4096	SET TERMINATING CHARACTER REQUIRED
	CAM	9	
	CAM	8,M0	NEXT CHARACTER TO M8
	TRA	AF1	RETURN TO FORMAT TEST
BE5	CALL	READEC	INPUT NEXT NUMBER
	STC	5,1,	STORE MOST SIGNIFICANT HALF
	CRN	15,2	
	CAM	14	DOUBLE PRECISION BIT
	JPM	14, BE6	BRANCH IF SINGLE PRECISION
	STR	5,1,	STORE LEAST SIGNIFICANT HALF
	TRA	BE6	GO TO END TESTS
BE4	CAM	3, BE7	
	JPM	4, BE9	
	ATN	1,1,	GET NEXT IO CONTROL WORD
	LFR	5	
	TRA	BE7	
BSTAR	CAM	14, PUSHDN+1-M13	ASTERISK, CHECK FOR EMPTY PUSHDOWN
	JUM	14, FRMERR	STILL INSIDE PARENTHESIS
	CAM	13, PUSHDN+2	SET PUSHDOWN TO LAST LEFT PARENTHESIS (OUTSIDE)
	LDM	9, M13	FORMAT ADDRESS OF LAST LEFT PARENTHESIS
	JUM	9, BSTAR3	ZERO IF NO PARENTHESES ENCOUNTERED
	CAM	13, PUSHDN+1	SET PUSHDOWN TO BEGINING OF FORMAT
BSTAR3	LDM	11, M13	RESET PARENTHESIS REPEAT COUNT
	FIL		
RESTAR	CAM	3, BSTAR1	
	JUM	6, FRESET	NEW LINE ON RETURN AND GO TO BACK UP
	JPM	4, BSTAR4	LAST CONTROL WORD
	ATN	1,1,	
	LFR	5	LOAD NEXT CONTROL WORD
	TRA	RESTAR	
BSTAR4	CAM	3, RESTAR	
BE9	CRM	4, 12	
	JNM	4, BE9A	REPEAT ENTRY BIT ON
	CRN	15, 10	
	CAM	14	

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 49 of 62
 Change:

	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, ,
	CAM	2
	JNM	7, BS4
BS5	CALL	FLDBF
	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
RESTORE 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	50 of 62
Change:	

	CJU	2,BS4	
	TRA	BS3	
BH	SFN	9,,	HOLLERITH FIELD, INVERT COUNT
	CAM	2	
	JZM	9,BH6	ZERO COUNT
	CRN	15,10	
	CAM	14	INPUT OUTPUT BIT
	JNM	14,BH2	IMPUR
BH1	CALL	FORMRD	READ FORMAT CHARACTER
	CALL	FLDBF	OUTPUT
	CJU	2,BH1	COUNT
	TRA	BH6	
BH2	SFR	5,FCOM12	SAVE F5
	LEF	5,FRCNT	FORMAT CONTROL WORD
	SFR	7,FCOM13	SAVE F7
BH5	CALL	FRDBF	READ CHARACTER
	CAM	9,63	SET MASK
	CJZ	4,BH3	LEFT OR RIGHT. BRANCH IF RIGHT
	CRM	8,7	MOVE TO LEFT
	CRM	9,7	DITTO MASK
BH3	LEF	7,M7	CURRENT FORMAT WORD
	JLH	BH3A+M6+M6+8	BRANCH ACCORDING TO QUARTER WORD
	FIL		
BH3A	NAM	12,M9	FIRST QUARTER
	ORM	12,M8	CHARACTER REPLACED IN FORMAT
	JZM	4,BH4	RIGHT HAND PART
	TRA	BH3B	
	NAM	13,M9	SECOND QUARTER DITTO
	ORM	13,M8	
	JZM	4,BH4	
	TRA	BH3B	
	NAM	14,M9	THIRD QUARTER DITTO
	ORM	14,M8	
	JZM	4,BH4	
	TRA	BH3B	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	51 of 62
Change:	

	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
	CRM	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

STROE 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. RESTROE FORMAT COUNT, F5 AND F7

THIS IS POSITIVE IF HOLLERITH, -VE IF A FIELD
 INCREMENT IO LIST ADDRESS
 DECREASE IO 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	52 of 62
Change:	

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	
	JLH	0,0	OTHERWISE EXIT
BSUB4	SFR	4,FCOM6	
	CALL	BINC	
	LFR	4,FCOM6	
	FIL		
RENN	JUM	6,RENN	
	CAM	3,RENN	
	JPM	4,BE9	
REN1	CAD	5,1,	
	ADM	6,-1	
	SIA	8	
	CAM	9,M8	
	TRA	BSUB	
FLDBF	SFR	5,FCOM7	SUBROUTINE WHICH PUTS M8 IN OUTPUT BUFFER AS NEXT CHARACTER
	SFR	6,FCOM8	
	SFR	7,FCOM9	
	LFR	7,FBFWD	BUFFER PEPARATION WORD
	LFR	5,FCNTS	CONTROL WORD
	JPM	4,FERRPR	TOO MANY CHARACTERS FOR LINE OR CARD
	JLH	7,0	BRANCH ACCORDING TO QUARTER
	FIL		FIRST QUARTER, CHAR. TO M12
M	ADM	12,M8	BRANCH IF RIGHT HAND PART
	CJZ	6,FN3A	MOVE TO LEFT
	CRM	12,7	
	TRA	FN8	SECOND QUARTER
	ADM	13,M8	
	CJZ	6,FN3A	
	CRM	13,7	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	53 of 62
Change:	

	TRA	FN8
	ADM	14, M8
	CJZ	6, FN3A
	CRM	14, 7
	TRA	FN8
	ADM	15, M8
	CJZ	6, FN6
	CRM	15, 7
	TRA	FN8
FN6	ATN	5, 1,
	SFR	7
	LFR	7, FZERO
	CAM	7, M-2
FN3A	CSM	6, 2
	ADM	7, 2
FN8	ADM	4, 1
FN8A	SFR	7, FBFWD
	SFR	5, FCNTS
	LFR	5, FCOM7
	LFR	6, FCOM8
	LFR	7, FCOM9
	JLH	3, 0
FBACK	SFR	4, FCOM12
	SFR	7, FT5
	LFR	7, M13
	LFR	4, FRCNT
	ANM	1, 8184
	ANN	13, 8184
	SBM	1
	CRN	1, 3
	SBM	3
	CAM	1, M13
	ANN	1, 1
	CAM	0, -2
	ANN	1, 6

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
 STROE 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	54 of 62
Change:	

CAM 2
 CRM 2,1
 SBM 2,4
 SFR 4,FRCNT
 LFR 7,FT5
 LFR 4,FCOM12
 JLH 3,0
 FORMRD SFR 4,FCOM12
 SFR 7,FCOM13
 LFR 4,FRCNT
 LFR 7,M3
 ORB 2,4
 CAM 8,M12
 CJZ 0,FORM2
 CRM 8,6
 FORM3 ADM 1,1
 ANM 8,63
 SFR 4,FRCNT
 LFR 4,FCOM12
 LFR 7,FCOM13
 JLH 3,0
 FORM2 CAM 0,-2
 CJU 2,FORM3
 CAM 2,-4
 ADM 3,1
 TRA FORM3
 FRESET ANN 15,512
 CAM 14
 SFR 5,FCOM7
 SFR 4,FCOM24
 SFR 7,FCOM9
 JUM 14,FNOOUT
 LFR 5,FCNTS
 FRST1 LFR 7,FBFWD
 TRA FBLNK2

QUARTER WORD COUNT RESET

STORE BACK IN MEMROY

RESTORE F5 AND F7

EXIT

SUBROUT+NE WHICH READS NEXT FORMAT CHARACTER

FORMAT CONTROL WORD

FORMAT WORD

EXTRACT QUARTER WORD

BRANCH IF RIGHT HAND PART

SHIFT

INCREMENT COUNT

EXTRACT 6 BITS

RESTORE CONTROL WORD

RESTORE F4 AND F7

EXIT

RESET LEFT RIGHT SWITCH

INCREMENT QUATER WORD COUNT

RESET QUARTER COUNT

INCREMENT ADDRESS OF FORMAT WORD

SUBROUTINE TO OUTPUT LINE OR CARD, OR INPUT A CA
RD DEPENDING ON SWITCHES IN M15

BRANCH IF INPUT

OUTPUT BUFFER CONTROL WORD

PREPARATION WORD TO F7

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 55 of 62
 Change:

FBLNK1	ATN	5, 1,	
	SFR	7	FILL OUT BUFFER WITH PREPARATION WORD, THEN
	LFR	7, FZERO	A
	ADM	4, 8	ALL BLANKS
FBLNK2	JNM	4, FBLNK1	MORE WORDS TO GO
	LFR	7, FCOM9	
	ANN	15, 256	
	CAM	14	PUNCH/PRINT BIT
	JUM	14, FN9A	BRANCH IF PRINT
	CALL	SYSIO	OUTPUT TO PUNCH VIA SYSTEM
	DECQ	WRITE+1, OUTBF, 0, 0	
	CAM	0, -80	RESET BUFFER CONTROL WORD FOR PUCH
	CAM	2, -2	
	CAM	3, M	
	TRA	FN9B	
FN9A	CALL	SYSIO	PRINT VIA SYSTEM PROGRAM
	DECQ	WRITE+2, OUTBF, 0, 0	
	CAM	0, -133	RESET BUFFER CONTROL WORD FOR PRINT
	CAM	2, -1	
	CAM	3, 2+M	
FN9B	CAM	1, OUTBF	BUFFER ADDRESS RESET
	SFR	4, FCNTS	STORE FORU BACK INTO BUFFER CONTROL
	LFR	7, FZERO	
	SFR	7, FBFWD	RESTROE F7
	TRA	FRSTNC	
FNOOUT	CALL	SYSIO	INPUT CARD SECTION
	DECQ	1, INPBF, 0, 0	
	CAM	0, -80	RESET CARD READ CONTROL WORD
	CAM	2, -2	
	CAM	3, -4	
	CAM	1, INPBF	
	SFR	4, FCNTSC	STORE IN CARD READ CONTROL
FRSTNC	LFR	5, FCOM7	RESTORE F4, F5 AND F7
	LFR	4, FCOM24	
	LFR	7, FCOM9	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	56 of 62
Change:	

	JLH	3,0
FRDBF	SFR	5,FCOM7
	SFR	7,FCOM9
	LFR	5,FCNTSC
	JPM	4,FERRPR
	LFR	7,M5
	ADM	4,1
	ORB	7,4
	CAM	8,M12
	CJZ	6,FNC2
	CRM	8,6
FNC3	ANM	8,63
	SFR	5,FCNTSC
	LFR	5,FCOM7
	LFR	7,FCOM9
FNC2	JLH	3,0
	CAM	6,-2
	CJU	7,FNC3
	CAM	7,-4
	ADM	5,1
	TRA	FNC3
BX	CAM	10,M9-1
	CAM	9,1
	CAM	0,27
	TRA	BX1
BA	CAM	10,M9-1
	CAM	9
	JSB	0,BSUB
	FIL	
	CAM	14,M8+11
	SFR	7,FCOM5
	LFR	7,FRCNT
	SFR	7,FCOM6
	CAM	0,M9
BA4	LFR	7,FCOM5

EXIT
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
STROBE CONTROL WORD BACK
RESTORE F5 AND F7

EXIT
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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	57 of 62
Change:	

	CALL	BINC
	FIL	
REBA	JZM	6, BA2
	CAM	12, -2
	CAM	14, -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,
	LFR	5
	TRA	BA4
BM	ADM	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 BOUNDRARY

CONSTRUCT FAKE FORMAT CONTROL WORD

RESTROE F7

BRANCH IF LESS THAN 8 CHARACTARS LEFT
 SET COUNT TO 8 CHARACTERS
 SET MARKER IN M7 SO THAT CORRECT EXIT IS MADE FR
 M HOOLERITH

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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	58 of 62
Change:	

SFR 7,FCOM6
 JPM 11,BQ4A
 JNM 0,BQ1
 CALL READEC
 SIA 8
 JZM 0,BL1
 BQ4 LFR 7,M5
 JPM 11,BDOUT
 ANN 4,3
 TRA BD4
 BD4 CAM 12,M8
 TRA BD5
 CAM 13,M8
 TRA BD5
 CAM 14,M8
 TRA BD5
 CAM 15,M8
 BD5 SFR 7,M5
 BD11 ANN 4,3
 CAM 3,-3
 ADM 4,1
 JUM 3,BD7
 ADM 4,-4
 BD6 ADM 5,1
 BD7 ADM 6,-1
 LFR 7,FCOM6
 ADM 10,-1
 JPM 10,BD3
 LFR 6,FCOM5
 CAM 0,M8
 TRA BS7
 BD2 CAM 3,BD3
 JPM 4,BE9
 ATN 1,1,
 LFR 5

SAVE F7
 BRANCH IF OUTPUT
 BRANCH IF OCTAL
 READ DECIMAL NUMBER
 STORE AS INTEGER IN M8
 BRANCH IF L FIELD
 DATA WORD TO M8
 BRANCJ IF OUTPUT
 BRANCH ACCORDING TO QUARTER WORD
 FIRST QUARTER
 SECOND QUARTER
 THID QUARTER
 FOURTH QUARTER
 STORE BACK IN MEMORY
 INCREASE Q
 NOT WORD BOUNDARY
 INCREASE WORD COUNT
 DECREASE ITEM COUNT
 RESTORE F7
 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

Date: 7/16/64
 Section: 8.4-M2-PRINT
 Page: 59 of 62
 Change:

	TRA	BD3
BQ4A	JZM	0, BM1
	CJU	0, BQ4
	CAM	0, -1
BM1	CAM	7, M5
	CAM	10
	TRA	BOUT1
BDOU7	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
BOUT4	CALL	FLDBF
	ADM	9, -1
BOUT3	JPM	9, BOUT4
	CAM	11, -5
	CRM	7, 12
	ANN	7, 1
	CAM	8
	TRA	BOUT5
BOUT7	CRM	7, 10
	ANN	7, 7
	CAM	8
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 M7
CLEAR REPEAT COUNT

EXTRACT QUARTER WORD FOR PRINTING
OCTAL PRINT
NUMBER TO ACCUMULATOR

STE 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

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	60 of 62
Change:	

BOUT5Z	CALL	FLDBF	OUTPUT CHARACTER
	CAM	9,-1	STT BLANK INHIBIT
BOUT6	CJU	11,BOUT7	COUNT LENGTH OF FIELD
	CJU	0,BD11	NOT ADDRESS PRINT
	TRA	BD7	
BQ1	CSM	9,M2	OCTAL READ PROGRAM
	CAM	7	CLEAR ASSEMBLY AREA
BQ2	CALL	FRDBF	READ CHARACTER
	JZM	8,BQ3	IGNORE BLANK
	CRM	7,10	SHIFT ASSEMBLED WORD LEFTT 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	COUNT CHARACTERS
	CAM	8,M7	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,-4	
FCNTS	DECQ	-133,OUTBF,-2,M	
FZERO	DECQ	???	
OUTBF	BSS	17	
INPBF	BSS	10	
PUSHDN	BSS	5	
	ASSIGN	FCOM1,FCOM2,FCOM3,FCOM4	
	ASSIGN	FRCNT,FCOM12,FCOM13	
ERROR	CAM	1,ERM1	
	TRA	ERPR	
ERRORT	CAM	1,ERM2	
	TRA	ERPR	
FERRPR	CAM	1,ERM3	

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	61 of 62
Change:	

	TRA	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	DECQ	3,EMES,1,MES1
ERM2	DECQ	3,EMES,1,MES2
ERM3	DECQ	3,EMES,1,MES3
ERM5	DECQ	3,EMES+1,1,MES5
MES1	CHR	32,26H DATA CHARACTER INCORRECT Q5*
MES2	CHR	24,17H FIELD TOO SMALL Q5*
MES3	CHR	40,29H TOO MANY CHARACTERS ON LINE Q5*
MES5	CHR	24,14H FORMAT ERROR Q5*
	ASSIGN	FCOM7,FCOM8,FCOM9
	ASSIGN	FCOM6,FCOM5,FCOM24
	ASSIGN	FBFWD,FT5,FT6
WRITE	EQU	512

Date:	7/16/64
Section:	8.4-M2-PRINT
Page:	62 of 62
Change:	

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

Date:	3/5/63
Section:	8.4
Page:	1 of 1
Change:	