File No. 1800-36
Form No. C26-5990-1

Systems Reference Library

IBM 1800 Time-Sharing Executive System Specifications

This publication describes and illustrates the use of the IBM 1800 Time-Sharing Executive System. The calling sequences to all the programs are described in terms of the FORTRAN CALL statements. The Assembler language equivalents of the calling sequences are included in an appendix.

It is assumed the reader has knowledge of the 1800 Data Acquisition and Control System and the 1800 FORTRAN Language.

The 1800 Time-Sharing Executive System provides the user with an easy means of generating, testing, and executing programs for process control and data acquisition applications as well as nonprocess applications.

PREFACE

This publication presents the specifications for the IBM 1800 Time-Sharing Executive (TSX) System. The 1800 TSX system provides a means for generating, organizing, testing, and executing programs for process control and data acquisition applications and nonprocess operations.

The publication is divided into four parts:

- Introduction
- System Director
- Nonprocess Monitor
- Subroutines

The first section describes the terms used, the layout of core and disk storage, and general system operation.

The next three sections describe the operation, statements, and control records of the system programs. Linkages to the various system programs are described in terms of the FORTRAN CALL statements used. The equivalent calling sequences for 1800 assembler language programs are given in Appendix A.

The FORTRAN language and assembler language for the system are described in the publications listed below. Also listed are the publications describing the IBM 1800 Data Acquisition and Control System. To fully understand the material in this manual, the reader should be familiar with the information contained in these publications.

IBM 1800 FORTRAN Language (Form C26-5905)
IBM 1800 Assembler Language (Form C26-5882)
IBM 1130/1800 Plotter Subroutines (Form C26-3755)
IBM 1800 Data Acquisition and Control System Functional Characteristics (Form A26-5918)
IBM 1800 Data Acquisition and Control System Data Processing Input/Output Units (Form A26-5969)

MACHINE REQUIREMENTS

The machine units required for operation of the 1800 TSX System are listed below. Normally, only one device per data channel is permitted. For exceptions to this rule see Test Function under BASIC CALLING SEQUENCE in the Subroutine Library section.

1. IBM 1801 or 1802 Processor-Controller, with a minimum of 8192 words of core storage.
2. IBM 2310 Disk Storage.
3. IBM 1053 Printer or IBM 1443 Printer or IBM 1816 Printer-Keyboard.
4. IBM 1442 Card Read Punch.

MACHINES AND FEATURES SUPPORTED

In addition to the machine units required, the following optional units and features can be used during operation of the TSX System.

1. Additional core storage (total, 32, 768)
2. Additional IBM 2310 Disk Storage units (total, 3)
3. Additional IBM 1442 Card Read Punch unit (total, 2)
4. Additional IBM 1816 Printer-Keyboard unit (total, 2)
5. Additional IBM 1053 Printer units (total of eight 1053s and 1816s)
6. Additional Data Channels (total, 9)
7. Additional Interrupt Levels (total, 24)
8. Multiplexer Unit (S and R)
9. Analog-Digital Converter (total, 2)
10. Digital/Analog Output
11. Digital Input
12. Comparator
13. IBM 1443 Printer unit (total, 1)
14. IBM 2401-2402 Magnetic Tape Units (total, 2)
15. IBM 1627 Plotter unit (total, 1)
16. IBM 1054 Paper Tape Reader.
17. IBM 1055 Paper Tape Punch.

This edition is a major revision of the previous edition (C26-5990-0), which is now obsolete. Significant changes have been made throughout this edition; therefore, it should be reviewed in its entirety.

CONTENTS

The IBM 1800 Time-Sharing Executive (TSX) System is a group of programs designed to provide the user with a means of generating, organizing, testing, and executing user-written programs for process control, data acquisition, and non-process-control applications. The user-written programs can be written in FORTRAN language or symbolic form and compiled or assembled, tested, and stored on disk while the TSX System is monitoring the process under control. Nonprocess-control programs, such as scientific and engineering calculations, or normal data processing jobs, can be executed simultaneously with monitoring of the process. When a process interrupt is recognized, its occurrence is either recorded or control is transferred to the proper interrupt servicing program. When the interrupt servicing program has completed its operation, control is normally returned to the program that was operating when the interrupt occurred.

With the full complement of interrupt priority levels, it is possible for up to 28 interrupted programs to be in a suspended state of operation while the highest priority interrupt is being serviced by another program. Execution of each suspended program is resumed as the higher priority interrupt programs are completed.

The TSX system keeps track of the partially completed programs when interrupts occur and, when necessary, moves some of the programs to disk storage in order to provide a core storage area for the interrupt servicing routine. Through the use of various system-built tables, the TSX System can locate any required user's program or system program.

It is important to note that certain system programs must be assembled by the user when the system is received. This allows various program options and the user's particular machine configuration to be stated; thus, each user's system is tailored to fit his needs.

TERMS

The following terms are described briefly as they are used in this publication. More detailed definitions and descriptions are given in the applicable sections.

Process Program: A program that performs some function related to the process under control. It usually involves the use of analog/digital input/output devices. There are two types of process programs: interrupt and mainline.

Interrupt Program: A program that performs a desired operation due to the occurrence of an interrupt. The terms interrupt program, interrupt routine, and interrupt servicing routine are used synonymously.

Mainline Program: A program which does not directly service an interrupt (analysis programs, logging programs).

Nonprocess Program: A program that operates under control of the nonprocess monitor, but does not normally perform any function related to the process. It may be written by the user (instrument calibration, scientific or engineering calculations, payroll inventory) or provided by IBM (FORTRAN compiler, assembler).

Core Load: A complete, executable programming package, which is stored on the disk in core-image form. A core load consists of a main program (interrupt, mainline, or nonprocess), all required subroutines not permanently in core, and the core load communication tables. Mainline core loads can also include in-core interrupt routines.

System Skeleton: The permanently assigned area of core storage that contains the framework of the system, including the necessary programs, work areas, communications areas for system operation, and any user defined options.

Variable Area: The area of core storage is reserved for process and nonprocess core loads, TSX programs such as the nonprocess monitor and system error programs.

Save: An operation in which a portion or all of variable core storage is moved to one of the save areas on disk to make room for a higher priority operation.

Restore: An operation in which the contents of a disk save area are returned to the variable area of core storage.

Exchange: A save operation followed immediately by the overlaying of the variable core area with a new core load.

Time-Sharing: The ability to utilize the P-C (processor-controller) for execution of nonprocess programs during the time that process programs are not being executed. The P-C retains the ability to respond to process interrupts.

SYSTEM COMPONENTS

The 1800 Time-Sharing Executive system (Figure 1) consists of the system director, nonprocess monitor, and subroutine library.

- System Director - in general, handles all interrupts, controls the user-specified sequence of process control programs, and controls the time-sharing of nonprocess programs.
- Nonprocess Monitor - contains the supervisor program, disk utility program, FORTRAN compiler, assembler program, and the simulator program. It also controls operation of the user's nonprocess programs.
- Subroutine Library - contains the most often required subroutines used by process and non-process programs.

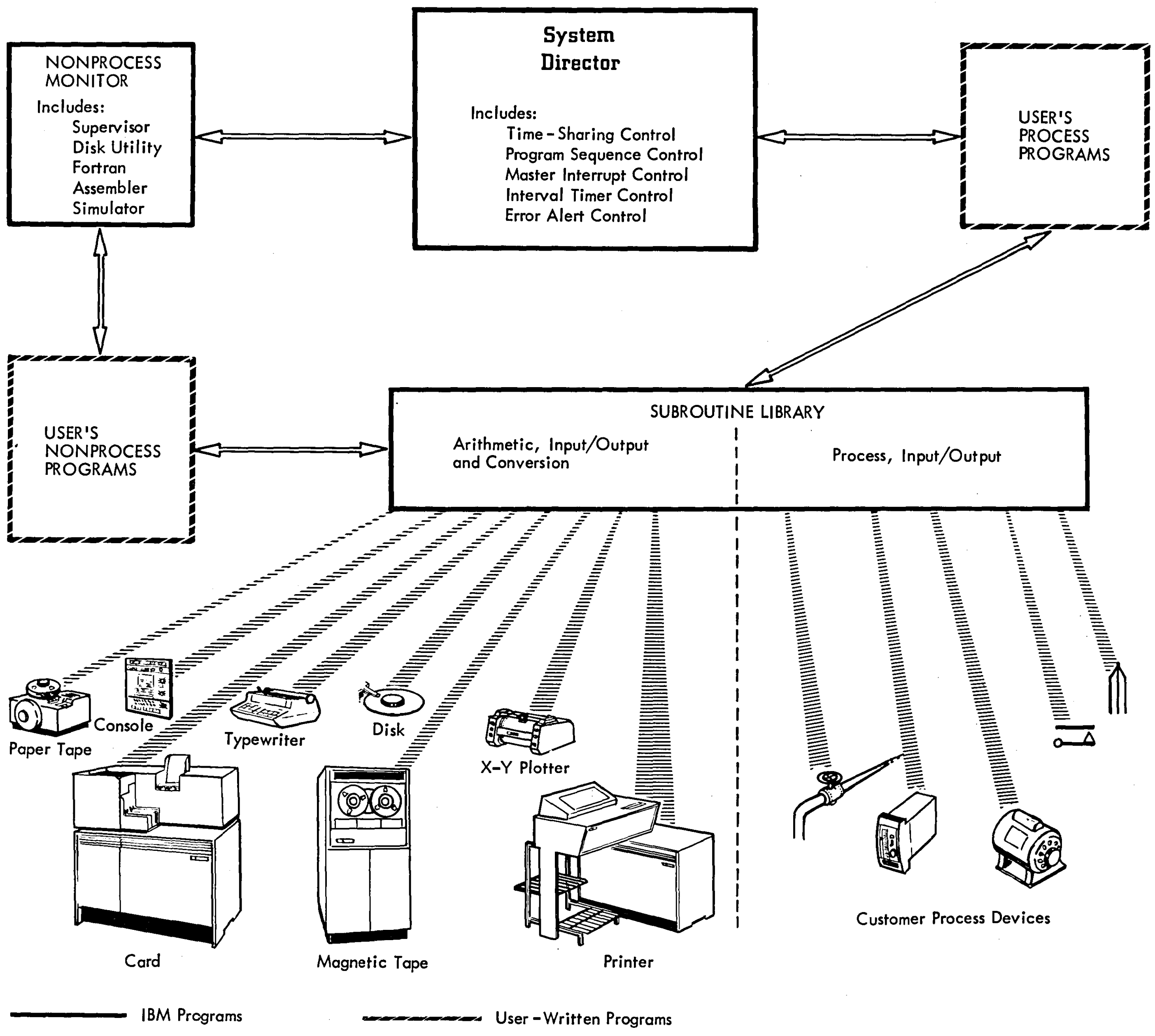


Figure 1. IBM 1800 Time-Sharing Executive System

In addition to these major components, the system includes loaders comprised of the system loader, skeleton builder, core load builder and temporary-assembled-skeleton. These programs are required for system generation and operation. The system loader and skeleton builder are used only for initial loading and system generation.

Disk Storage Areas

The areas of disk storage described below are automatically assigned by the system, but vary in size depending on the system configuration and customer definition. Some of these areas can be modified, relocated, or removed, as shown in Figure 2.

IBM Systems Area. This area is for storage of the integral parts of the TSX System: the system director, nonprocess monitor, and the subroutine library. This area must be assigned to the pack on logical disk drive zero in multiple disk systems.

Core Load Area. This area is for storage of core image process and nonprocess core loads and data. All programs and data files stored in this area are assigned fixed disk locations. This permits the disk location of a process program to be kept with the calling program and results in faster access to the program. A core load area can be assigned to each disk drive.

Relocatable Program Area. This area is for storage of relocatable programs. The number of sectors required for a program to be stored is subtracted from the nonprocess work storage area if they are both assigned to the same disk pack. When a program is deleted, a separate DUP operation must be performed to repack the area. A relocatable program area can be assigned to each disk drive.

Nonprocess Work Storage Area. This area is used for temporary storage during the execution of nonprocess programs. It is used extensively during the operation of the nonprocess monitor. For example, this area is used by the assembler during assembly of a source program and, at the conclusion of assembly, it contains the object program. In a FORTRAN nonprocess program, the DEFINE FILE statement can refer to this area. A nonprocess work storage area can be assigned to each disk drive.

Process Work Storage Area. This area is used for temporary storage during execution of process programs. In a FORTRAN process program, the DEFINE FILE statement refers to this area. A process work storage area can be assigned to each disk drive.

Save Areas. The save areas are used for temporary storage of interrupted programs while other programs are utilizing core. The features or programs that require reservation of specific save areas are:

- Error alert control program (EAC). This area is always required and is used when the disk portion of the EAC program must be executed.
- Dump feature of EAC. This area is required if the core dump option is specified in the user's error subroutine.
- Special calls. This area is required if any of the user's programs contain the CALL SPECL statement and no time shared operations are done. When time sharing is specified, the special calls area is not required unless time sharing is entered from a core load that was entered with a CALL SPECL.
- Interrupt core loads. This area is required only if any process interrupts are serviced by interrupt core loads and if the time-sharing feature is not utilized (see next item if time sharing is utilized).
- Time-sharing. Two save areas are required if the time-sharing function is utilized; one each for the saving of process and nonprocess core loads.
- FORTRAN I/O. A sector for each interrupt priority level that uses FORTRAN I/O must be provided to save FORTRAN I/O data and parameters when that routine is re-entered.

Any combination from the minimum of one to the maximum of seven save areas can be included. The save areas can be on any drive but the process save area and the special calls area must be on the same drive, and the dump area and EAC save area must be on the same drive.

Location Equivalence Table (LET). The location equivalence table serves functionally as a "map" for IBM subroutines and relocatable programs. Every relocatable program or subroutine stored on disk has at least one entry in the table. A table entry contains the name of the item and location information. Each entry point in a subroutine requires one entry in LET. A LET is required for each disk pack.

All operations which involve adding and/or deleting relocatable programs make reference to LET. (Core loads and data files refer to FLET.)

Fixed Location Equivalence Table (FLET). The fixed location equivalence table serves as a "map" for the location of core loads and data files. Each core load or data file has at least one entry in FLET.

A FLET is required for each disk pack.

Core Storage Layout

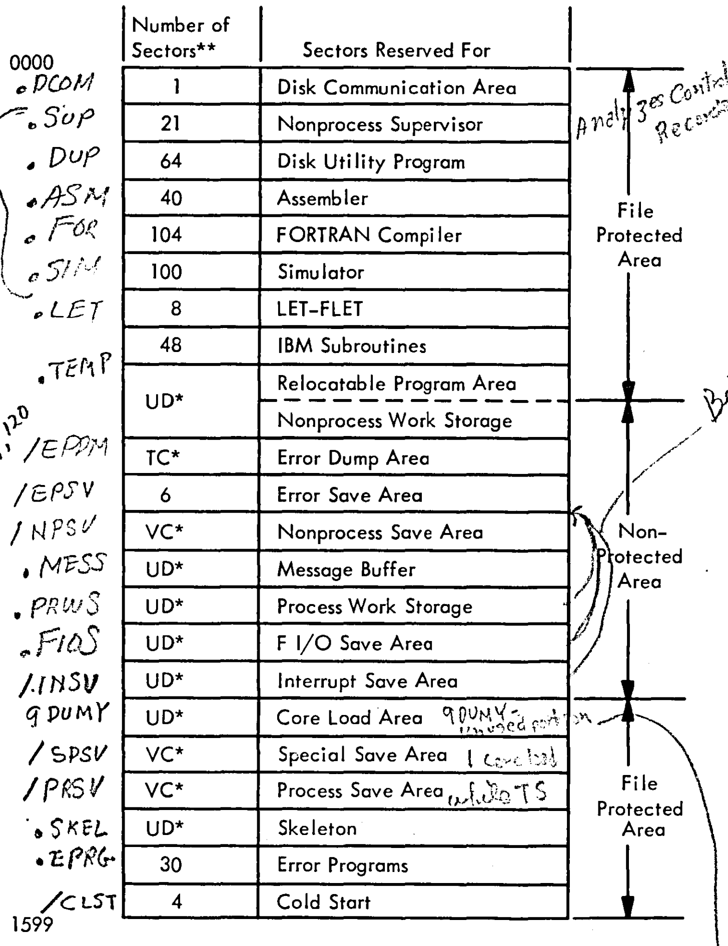
The layout of core storage depends upon the amount of core storage available. For a system with 8192 words (minimum core size for TSX), the fixed and variable areas of core storage are approximately equal in size.

The fixed area (termed the system skeleton) is used for the system director, disk input/output routine, printer routines, the skeleton COMMON area, skeleton program name table, executive branch table, and skeleton interrupt branch table. It can also contain any desired library or user-written subroutine.

The variable area is used for operation of the nonprocess monitor, and process, interrupt, or nonprocess core loads. This area, shown in Figure 3, cannot be less than approximately 3700 words in size if any time-sharing operations are to be executed.

Common Areas. There are three areas of core storage that can be used for FORTRAN COMMON storage: one located within the system skeleton, one located at the high-address end of core storage, and one located at the high-address end of the interrupt area.

The skeleton COMMON area can be referenced by any process or nonprocess program. To assign a



	Number of Sectors**	Sectors Reserved For	
0000 .DCOM	1	Disk Communication Area	File Protected Area
.SUP	21	Nonprocess Supervisor	
.DUP	64	Disk Utility Program	
.ASM	40	Assembler	
.FOR	104	FORTRAN Compiler	
.SIM	100	Simulator	
.LET	8	LET-FLET	
	48	IBM Subroutines	
.TEMP	UD*	Relocatable Program Area	
		Nonprocess Work Storage	Non-Protected Area
119,120 /EPDM	TC*	Error Dump Area	
/EPSV	6	Error Save Area	
/NPSV	VC*	Nonprocess Save Area	
.MESS	UD*	Message Buffer	
.PRWS	UD*	Process Work Storage	
.FIOS	UD*	F I/O Save Area	
/.INSV	UD*	Interrupt Save Area	
9DUMY	UD*	Core Load Area 9DUMY - unused portion	File Protected Area
/SPSV	VC*	Special Save Area 1 core load	
/PRSV	VC*	Process Save Area w/o TS	
.SKEL	UD*	Skeleton	
.EPRG	30	Error Programs	
/CLST 1599	4	Cold Start	

Analyzes Control Records

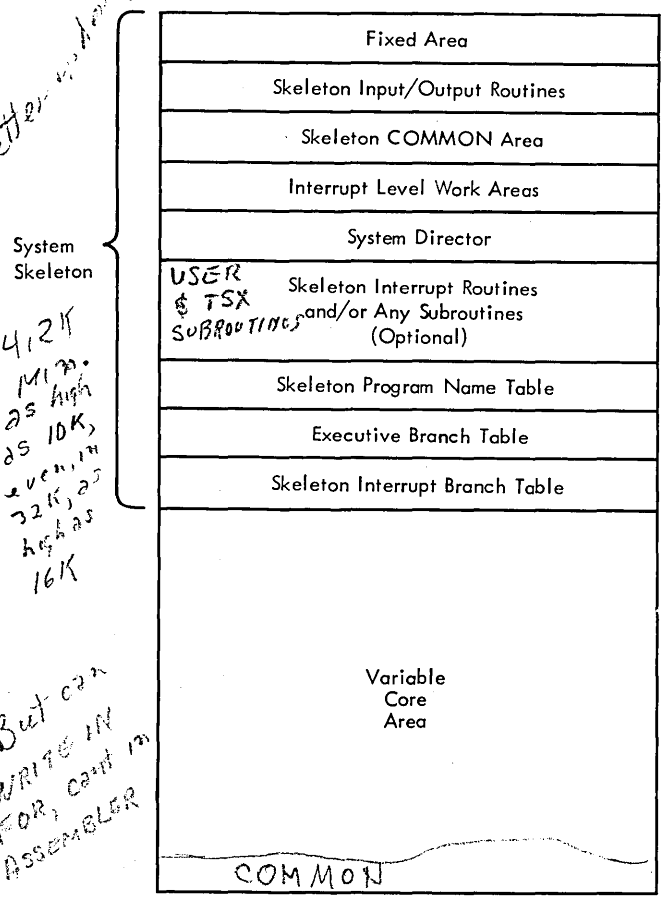
Better move to keep .MESS & 9DUMY together

But can WRITE IN FOR, can't in ASSEMBLER

*NOTE: TC indicates enough sectors to store total core; if error dump is not used, no space is reserved. VC indicates enough sectors to store variable core. UD indicates user defined; relocatable program area boundary increases or decreases nonprocess work storage as programs are deleted or added, respectively.

**NOTE: Sector quantities are approximate figures.

Figure 2. Example of Disk Layout


Fixed Area
Skeleton Input/Output Routines
Skeleton COMMON Area
Interrupt Level Work Areas
System Director
System Skeleton
USER $ TSX SUBROUTINES
Skeleton Interrupt Routines and/or Any Subroutines (Optional)
Skeleton Program Name Table
Executive Branch Table
Skeleton Interrupt Branch Table
4,2K Min. as high as 10K, as even in 32K, as high as 16K
Variable Core Area
COMMON

● Figure 3. Sample Core Storage Layout

variable to this area a special statement, COMMON/INSKEL, must be used. All other attributes of the COMMON statement remain the same. This area must be used for communications between: PROC ON NPROC PROC & INT

1. Core loads of a different type
2. Interrupt core loads
3. Combination core loads (if either is executed as an interrupt core load).
4. A special core load and the mainline core load that calls it.
5. A mainline core load (that called a special core-load) and the core load that restores it.
6. A skeleton subroutine and any other subroutine or core load.

Skeleton subroutines must use COMMON/INSKEL/ statements only.

The COMMON area located at the high-address end of core storage can be referenced only by mainline or nonprocess core loads. The normal COMMON statement in a mainline, special, or nonprocess core load is used to refer to this area. This area is saved and restored when special core loads or time-sharing operations are initiated or terminated; i.e., communication between mainline core loads is possible and communication between nonprocess core loads is possible.

The third area for COMMON is used only for inter-program communication for programs that form an interrupt core load or, between combination core load when they are executed on the mainline level. The normal COMMON statement in an interrupt or combination core load is used to refer to this area. The highest addressed location of this area must be assigned by the user when the system is assembled. This assigned location is the high-address boundary of the variable core storage area that is saved when an interrupt core load is loaded for execution. Thus, it is necessary to save only the area specified by the user for interrupt core loads (not the whole variable area).

CORE LOADS

Normally, core storage is not large enough to contain all process control programs at one time; therefore, these programs must be formed into smaller units which are termed core loads (Figure 4). A core load consists of a mainline program or interrupt program, in-core interrupt routines, and all required subroutines that are not with the system skeleton.

Core loads are stored on disk in core-image form to facilitate rapid loading when the core load is called for operation. Core loads are formed (and loaded to disk) by the disk utility program and the core load builder program.

Transfer Vector (TV). The core load transfer vector is divided into two sections: the executive section, which refers to the subroutines located within the system skeleton, and the variable section, which refers to the subroutines in the variable area of core. The TV area serves as a link between programs and subroutines. There is at least one three-word entry in the TV for each LIBF subroutine called by a program.

Interrupt Status Table. This table is also included as part of the header information for each core load. It specifies which interrupt routines are in core with this mainline core load and gives the entry address for the interrupt routine. Interrupts that are to be recorded for servicing at a later time are also specified (see Master Interrupt Control Program).

Program Name Table. The name table is part of the header information placed with each core load. It consists of the name and disk address of any other core loads that are called by this core load, as well as the name of a core load specified for restart (see Error Alert Control Program).

LOCAL Subprograms. LOCAL subprograms are read into core storage for execution when called by the object program. All LOCALs associated with the same program use the same area of core storage by overlaying one another as they are called. A copy of each LOCAL subprogram used with a core load is kept on disk in core-image format, together with the core load.

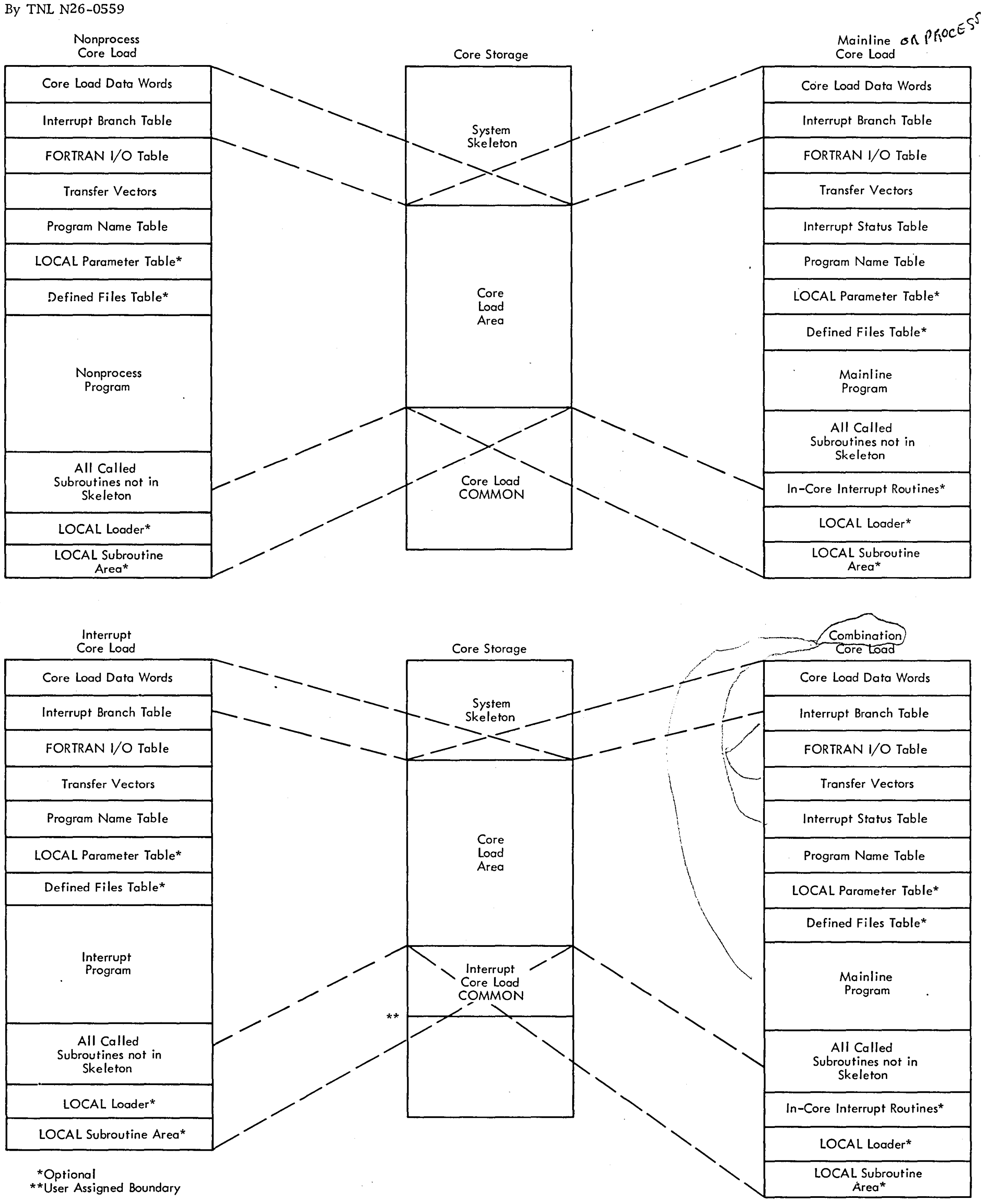


Figure 4. Sample Core Loads

The system director forms the heart of the TSX. It remains in core storage at all times, and all permanent areas are storage protected to ensure that they are not accidentally altered. Control is transferred to the system director as a result of any of the following:

1. TSX CALL statements in the user's program.
2. Interrupts.
3. Errors.

Basically, the system director is made up of five control programs and two data areas.

Control Programs

- Program Sequence Control (PSC) - controls the sequencing and initiates the loading and execution of user-specified process core loads.
- Master Interrupt Control (MIC) - automatically determines the type of each interrupt as it is recognized and transfers control to the proper interrupt servicing routine.
- Interval Timer Control (ITC) - provides a programmed real-time clock, a timer for TSC, nine programmed interval timers, and control of two machine-interval timers.
- Time-Sharing Control (TSC)* - controls the time-sharing of variable core between process and nonprocess core loads.
- Error Alert Control (EAC) - provides the following functions whenever an error occurs: (1) optionally saves core for future reference, (2) optionally branches to a user's program for further error analysis, (3) prints an error message, and (4) executes a specified recovery procedure.

Data Areas

- Mainline Core Load Queue Table - contains the names of mainline core loads (and their respective priorities) that have been queued for future execution.

*If the time-sharing function is not desired, the TSC program can be omitted.

- Level Work Areas - contains interrupt level instructions, MIC linkages, and work areas. One level work area is required for:

1. Each interrupt level utilized
2. Process mainlines
3. Nonprocess core loads
4. Trace and CE level
5. Internal error level

PROGRAM SEQUENCE CONTROL PROGRAM

Program sequence control (PSC) is a control program that handles the flow of control from one mainline core load to the next. PSC functions are initiated by execution of PSC CALL statements in the user's program. The specific functions of PSC are:

1. Execute the next sequential mainline core load. The new core load overlays the one that contained the call.
2. Save the mainline core load in progress (on disk) and load a special core load for execution.
3. Restore the core load that was saved in item 2 and continue execution from where it left off (the statement following the CALL SPECL).
4. Queue mainline/core loads associated with interrupts whose occurrence has been recorded.
5. Execute the highest priority mainline core load listed in the core load queue.
6. Insert mainline core load entries into or delete them from the core load queue.

For PSC to perform the above functions, a CALL statement must be executed for each one. The specific CALL statements and their parameters are described below.

Functionally, the CALL statements are divided into two groups: those for direct sequence, in which one mainline core load calls another, and those for queuing, in which either the highest priority mainline core load named in the queue is called or the core load queue is modified by inserting or deleting an entry.

The CALL statements from both groups provide the user with the flexibility of implementing his unique scheduling requirements. Unless otherwise stated, the results are unpredictable if these CALL statements are used in a nonprocess program or if they are used incorrectly in process core loads.

DIRECT SEQUENCE STATEMENTS

The direct sequence statements are used to

- CALL the next mainline core load to be executed.
- Save the present mainline core load (on disk) and CALL a special mainline core load for execution.
- Restore and continue execution of the saved mainline core load.

These functions are performed by the PSC program resident in the System Director.

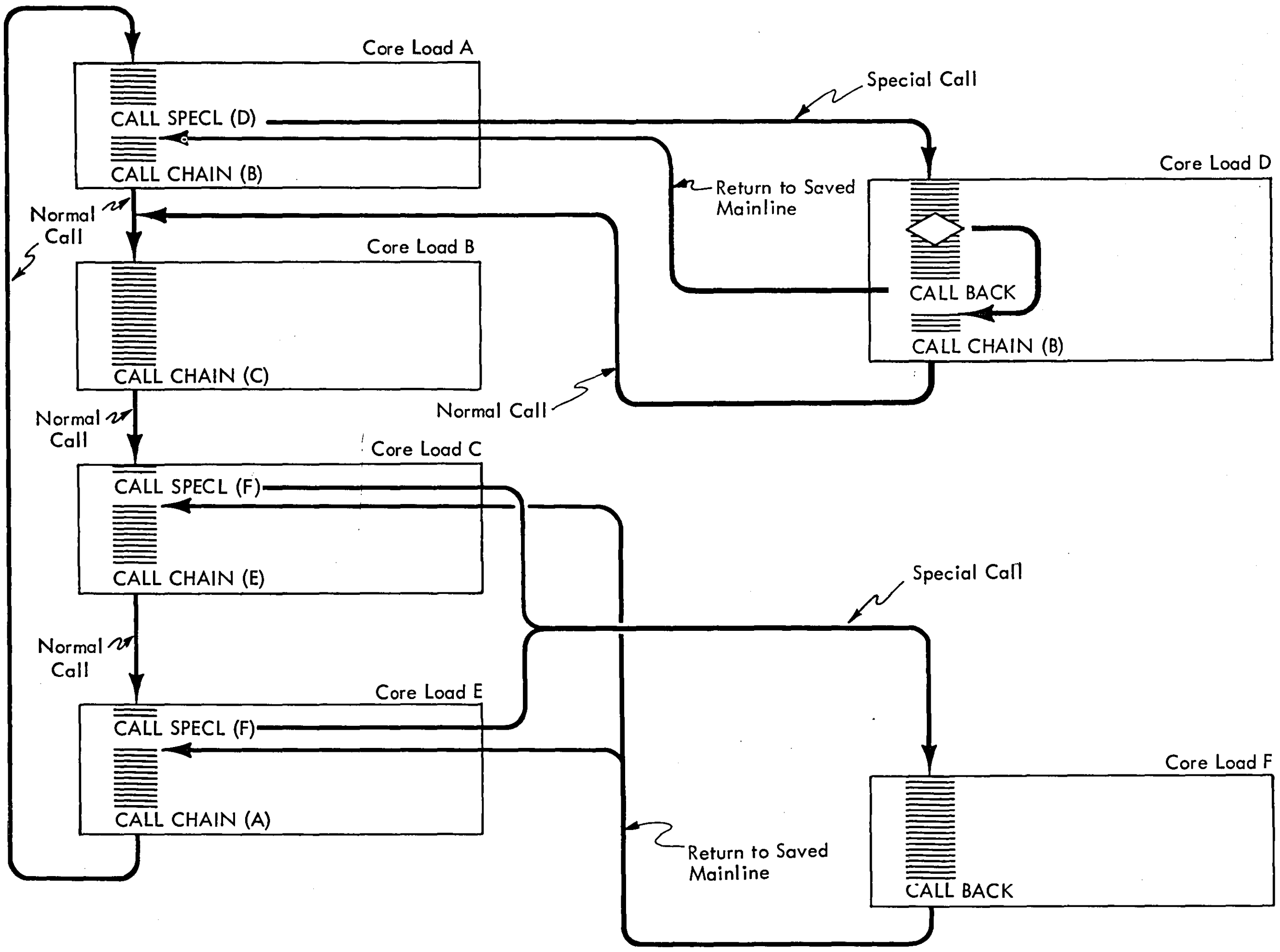
The specific CALL statements and their parameters are described in the following paragraphs and illustrated in Figure 5. These statements cannot be used in an interrupt program unless otherwise stated.

Core load names that appear in CALL statements must also be specified in a FORTRAN EXTERNAL statement. Core load names cannot be the name of a component program of a core load.

Normal Call - CALL CHAIN (NAME)

This call terminates execution of the mainline core load and transfers control to PSC, which loads the named mainline core load into core storage for operation. This is the last logical statement in a mainline core load; it calls the next mainline core load into operation.

NAME is the name of the mainline core load being called.



● Figure 5. Example of Direct Sequence Statements

Special Call - CALL SPECL (NAME)

This call suspends execution of the current mainline core load and transfers control to PSC, which

1. Saves the return address (i.e., the address of the instruction following the CALL SPECL statement).
2. Stores the current mainline core load on disk.
3. Loads and transfers control to the new (special) mainline core load.

NOTE: Only one mainline core load can be saved. Thus, if CALL SPECL is used in a core load that was called by a CALL SPECL, the mainline core load saved originally is lost.

Return Saved Mainline - CALL BACK

This statement is normally used as the last logical statement in a special mainline core load. When executed, it terminates the core load and transfers control to PSC which restores the last previously saved core load. Execution of the saved core load commences with the statement that follows the CALL SPECL statement.

CALL BACK is required only if the saved core load is to be restored and continued. Figure 5 shows the various direct sequence statements and the control path generated by each.

QUEUING STATEMENTS

The queuing statements are used to

- Insert entries into or delete entries from the core load queue.
- Execute the highest priority mainline core load specified in the queue.

These functions are performed by PSC subroutines, which can be located with either the calling program or the system director (user option).

The core load queue is a table that contains entries for mainline core loads that are to be executed. The table is located within the system director and is required only if either queuing techniques or recorded interrupts are used. Each entry in the queue table requires three words to specify name and priority (the priority requires one word, the name is converted to the word count and sector address which occupy one word each).

The maximum number of entries is specified by the user at system generation time. The specific CALL statements and their parameters are described in the following paragraphs.

Insert Into Queue - CALL QUEUE

This statement is used to place a mainline core load name and priority in the queue. If the same name and priority are already queued, they will not be placed in the queue a second time; however, the same name with a different priority can be inserted into the queue. The format of the statement is:

```
CALL QUEUE (NAME, P,E)
```

where

NAME is the name of a mainline core load that is to be entered into the queue.

P is the integer expression that specifies the execution priority for the core load. One (1) is the highest priority number. The allowable range of P is from 1 to ~~32,767.~~ 32767

E is an error parameter used to specify the action to be taken when the queue is full.

E = 0. Ignore call after printing an error message.

E = 1 through ~~32,766.~~ 32766. Replace the lowest priority entry in the queue with the name and priority in this call, if the priority of the queue entry is lower (numerically larger) than E. If there is no queue entry with a lower priority, the restart core load specified for this core load is executed.

E = ~~32,767.~~ 32767 Execute restart core load.

Different core loads can be assigned the same priority number, if desired. When two or more queue entries have been assigned the same priority, these entries have a priority among themselves on a first-in-first-out basis.

The CALL QUEUE statement can be used in any program. Examples in Figures 6, 7, and 8 provide a more complete description of the use of this statement.

Delete From the Queue - CALL UNQ (NAME, P)

This statement is used to delete a mainline core load entry from the queue. If the name and priority

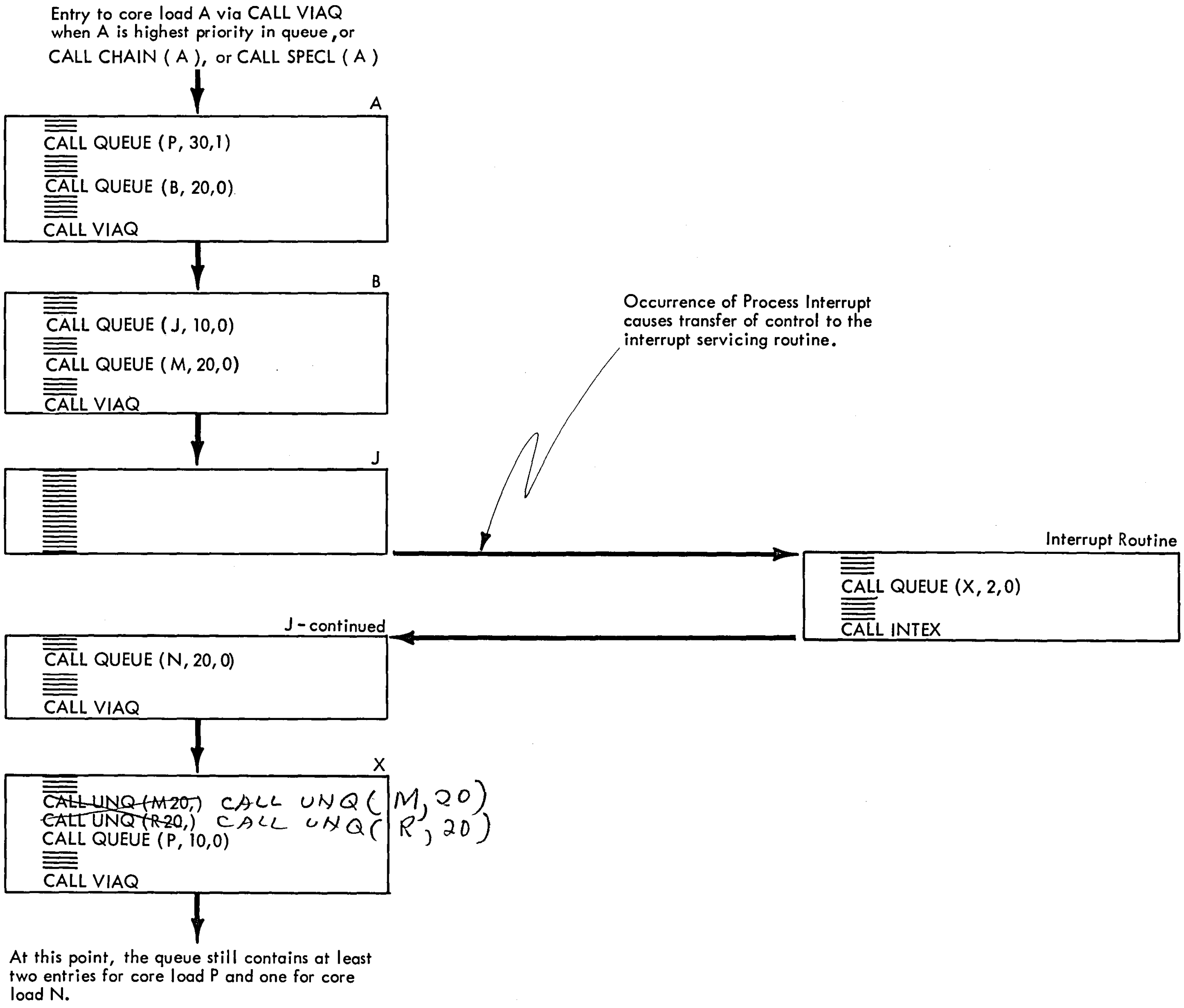

Entry to core load A via CALL VIAQ when A is highest priority in queue, or CALL CHAIN (A), or CALL SPECL (A)
A
CALL QUEUE (P, 30,1)
CALL QUEUE (B, 20,0)
CALL VIAQ
B
CALL QUEUE (J, 10,0)
CALL QUEUE (M, 20,0)
CALL VIAQ
J
Occurrence of Process Interrupt causes transfer of control to the interrupt servicing routine.
Interrupt Routine
CALL QUEUE (X, 2,0)
CALL INTEX
J - continued
CALL QUEUE (N, 20, 0)
CALL VIAQ
X
~~CALL UNQ (M20,)~~ CALL UNQ (M, 20)
~~CALL UNQ (R20,)~~ CALL UNQ (R, 20)
CALL QUEUE (P, 10,0)
CALL VIAQ
At this point, the queue still contains at least two entries for core load P and one for core load N.

Figure 6. Example of Queuing Statements

parameters do not match a queue entry, the statement has no effect. The CALL UNQ statement can be used in any program.

Execute Highest Priority Core Load - CALL VIAQ

This statement is used as the last logical statement in a mainline core load. It terminates the present core load and causes execution of the highest priority core load named in the queue.

When the CALL VIAQ statement is executed and there are entries in the queue, the highest priority entry is removed, ~~the remaining entries are compressed,~~ and ~~data contained in the removed entry is~~ used to call the core load it references.

If there are no entries in the queue, the process is considered to be in an idle condition (i.e., the process does not require any action at this time). Since variable core is not being utilized by process core loads, control is transferred to time-sharing control (TSC) for execution of nonprocess core loads. The time-sharing operation will continue for the period of time specified at assembly time or until terminated by an interrupt (see CALL ENDTS statement in the Time Sharing Control Program section). A CALL VIAQ operation is automatically performed when the time-sharing time is terminated. Therefore, if an interrupt program has placed a name in the queue, the named core load will then be automatically executed. (This is not true if time-sharing was initiated by a CALL SHARE statement.)

Problem: All programs of a given priority must be executed before a certain core load.

Solution:

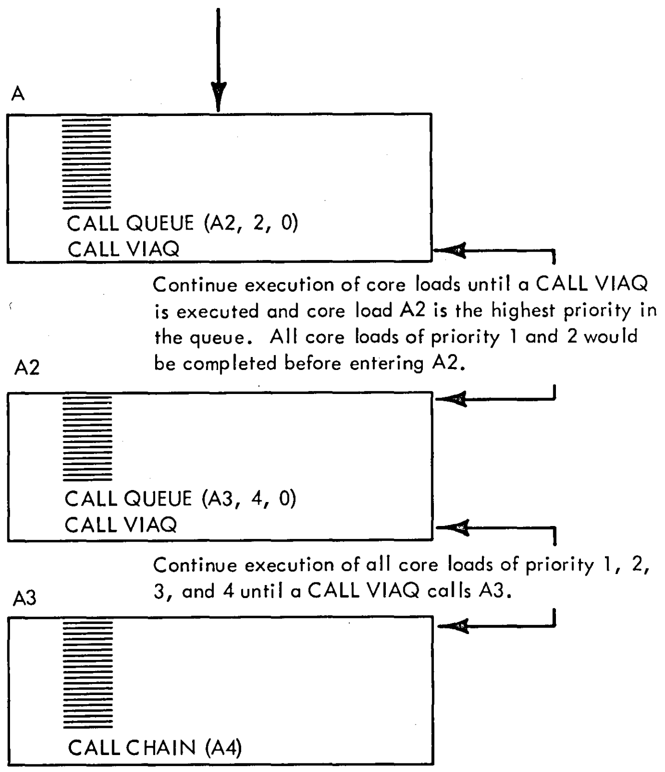

A
CALL QUEUE (A2, 2, 0)
CALL VIAQ
Continue execution of core loads until a CALL VIAQ is executed and core load A2 is the highest priority in the queue. All core loads of priority 1 and 2 would be completed before entering A2.
A2
CALL QUEUE (A3, 4, 0)
CALL VIAQ
Continue execution of all core loads of priority 1, 2, 3, and 4 until a CALL VIAQ calls A3.
A3
CALL CHAIN (A4)

Figure 7. Example of Core Load Sequences

The CALL VIAQ statement can be used only in mainline core loads. Figures 6, 7, and 8 are examples of core load sequences that can be obtained with the queuing statements.

Queue Core Load If Indicator Is ON - CALL QIFON

This statement is used to place a mainline core load name and priority in the queue table if its associated recorded interrupt indicator is on. Recorded interrupts are those that do not require service when they occur and can be recorded for servicing at a later time. Examples of the CALL QIFON statement are illustrated in Figure 9.

When an interrupt that is to be recorded is recognized by MIC, the interrupt is reset and a programmed indicator is set. It is the programmed indicators (set by MIC) that the QIFON subroutine interrogates. The statement format is:

CALL QIFON (NAME, P, L, I, E)

where

NAME is the name of a mainline core load.

P is the execution priority to be assigned to the mainline core load named.
L is the interrupt priority level or indicator (see L and I Combinations).
I is the PISW bit position indicator or CALL COUNT indicators (see L and I Combinations).
E is an error parameter used to specify the action to be taken when the queue is full.
E = 0. Ignore call after printing an error message.
E = 1 through 32,766. Replace the lowest priority entry in the queue with this call if the priority of the queue entry is lower than E. Restart if there is no entry lower than E.
E = 32,767. Execute restart core load.

L and I Combinations. The combination of L and I are

L	I	Reference
0-23	0-15	Process interrupts
0-23	-n	Programmed interrupts (see CALL LEVEL)
-n	0-31	Subprogram number for CALL COUNT statement (see Programmed Timers).

(-n means any minus number)

The CALL QIFON statement can be used in any process program.

Clear Recorded Interrupts - CALL CLEAR

The CALL CLEAR statement is used to clear the recorded interrupt indicators. In this way, specific interrupts or all external interrupts can be removed from their recorded status. The format of the statement is

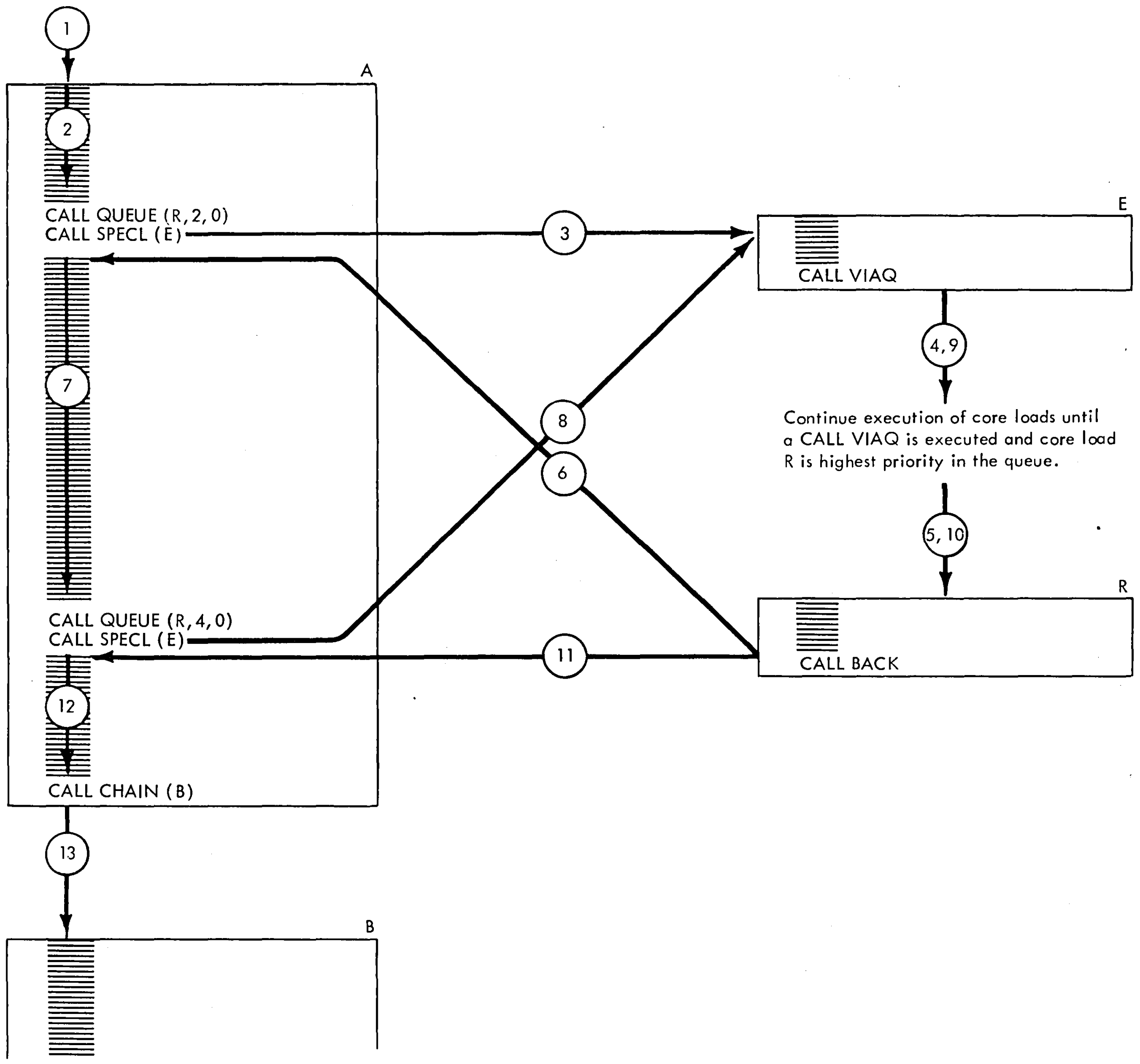
CALL CLEAR (M, L, I, L, I,...)

where

M is an integer constant that specifies the number of parameters to follow. If M equals 0, all indicators specifying recorded status are

Problem: Repeated execution of queued core loads during a given core load.

Solution: (The encircled numbers specify the sequence of operations.)



Note 1: The CALL SPECL statements cause core load A to be saved before transferring to core load E via lines 3 and 8. The CALL BACK statement in core load R causes core load A to be restored before the return is made via lines 6 or 11.

Note 2: Between lines 4 and 5 all core loads of priorities 1 and 2 will be executed; between lines 9 and 10 all core loads of priorities 1 through 4 will be executed.

Figure 8. Example of Core Load Sequences

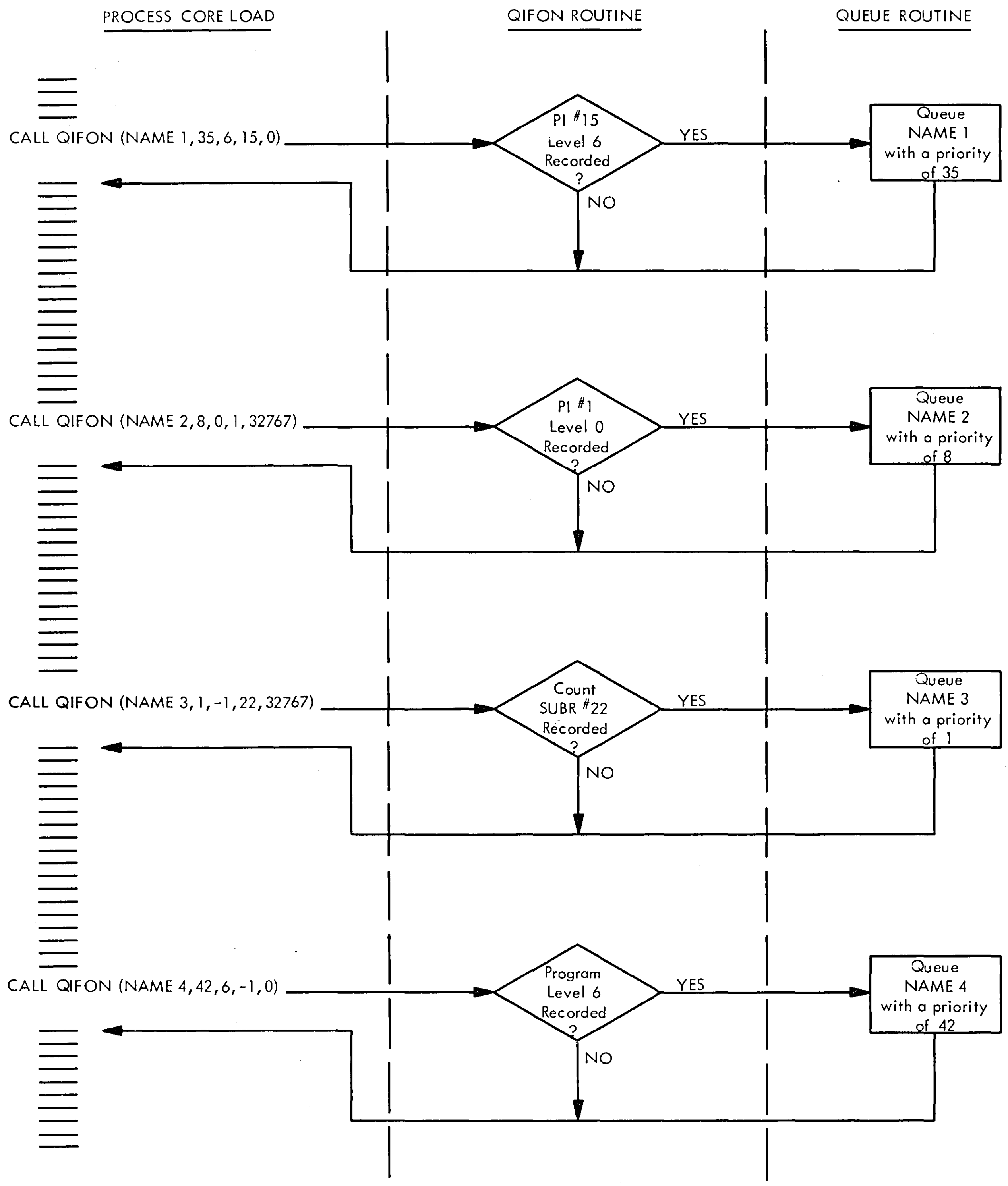

PROCESS CORE LOAD
QIFON ROUTINE
QUEUE ROUTINE
CALL QIFON (NAME 1,35,6,15,0)
PI #15 Level 6 Recorded ?
YES
NO
Queue NAME 1 with a priority of 35
CALL QIFON (NAME 2,8,0,1,32767)
PI #1 Level 0 Recorded ?
YES
NO
Queue NAME 2 with a priority of 8
CALL QIFON (NAME 3,1,-1,22,32767)
Count SUBR #22 Recorded ?
YES
NO
Queue NAME 3 with a priority of 1
CALL QIFON (NAME 4,42,6,-1,0)
Program Level 6 Recorded ?
YES
NO
Queue NAME 4 with a priority of 42

Figure 9. Example of CALL QIFON Statement

changed to indicate "not recorded".
L and I are the same as for the CALL QIFON statement.

The CALL CLEAR statement can be used in any process program.

MASTER INTERRUPT CONTROL PROGRAM

The master interrupt control (MIC) program controls the servicing of interrupts. An interrupt may occur at any time but it will not be recognized by MIC unless the interrupt is on a level that is not masked and is of a higher priority than the present level of machine operation. The user-specified assignment for interrupt levels determines the priority of a particular interrupt (see Appendix C). The user-assigned interrupts can be delayed from being recognized by masking the level to which they are assigned. The servicing of process, programmed, and COUNT subroutines can also be delayed by recording their occurrence. This function is explained under Recorded Interrupt Servicing.

Basically, there are two types of interrupts: internal and external. Internal interrupts are those associated with any input/output device, interval timer, trace, or error condition. Internal interrupts, except trace, are serviced by IBM-provided subroutines as soon as they are recognized.

EXTERNAL INTERRUPTS

External interrupts are those associated with the process and programmed interrupt features. They are serviced or recorded by one of four types of user-written routines: (1) Skeleton interrupt routine, (2) Mainline interrupt routine, (3) Interrupt core load, or (4) Mainline core load.

The different types of routines are provided to permit flexibility in the use of core storage, and in the response time requirements of a specific interrupt (i.e., the time required to enter an interrupt routine after the interrupt is recognized). Table 1 summarizes the major characteristics of the external interrupt servicing routines.

Interrupt routines are assigned to the skeleton area by control cards when the system skeleton is initially assembled. They are normally used to service process interrupts that require immediate response, have high priority, or that occur frequently.

Skeleton interrupt routines are required only if the user considers it necessary for the routine to always be in core storage (see Interrupt Assignment Restrictions).

External interrupts not serviced by skeleton interrupt routines can be serviced by routines included as part of a mainline core load. The response time of a mainline interrupt routine is the same as that of a skeleton interrupt routine only if the mainline core load containing the interrupt routine is in core when the interrupt occurs.

A mainline core load is required for the servicing of each external interrupt that might be recorded and serviced at a later time (see Recorded Interrupt Servicing). A mainline core load is not required for a recorded interrupt that is ignored or cleared at a later time.

Interrupt core loads are required for those interrupts that meet either of the following conditions.

1. User specifies the interrupt servicing routine to be out of core.
2. User specifies the interrupt servicing routine to be in core as part of a mainline core load.

If a time-sharing operation is in progress when an interrupt occurs, the interrupt (if not recorded) is serviced with the skeleton interrupt routine, if it exists, or with the interrupt core load. Even if the mainline that called for time-sharing has an interrupt routine for the interrupt that occurred, the interrupt core load associated with that interrupt is brought in (to core) for the servicing.

When recognized, external interrupts may either be recorded or serviced, as specified by the user. If recognition is recorded, it can be serviced later or cleared.

If not recorded, external process interrupts are serviced as soon as one of the following conditions becomes true.

1. The servicing routine is located within the system skeleton, the interrupt level is not masked, and an interrupt of higher priority is not being serviced.
2. No other external interrupt is being serviced, and the servicing routine is in core as part of the core load.
3. No other external interrupt is being serviced, the servicing routine is out of core, and no

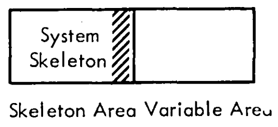
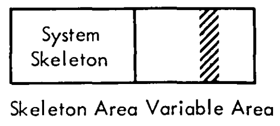
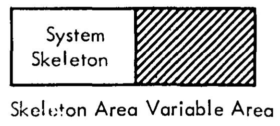
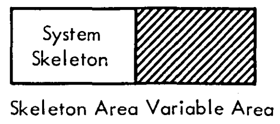
Type of Routine and Location	Characteristics
Skeleton Interrupt Routine Core Storage Location 	Permanently in core. Normally high priority. Can immediately interrupt lower priority routines. Fastest interrupt response. Must Call INTEX as last logical statement.
Mainline Interrupt Routine Core Storage Location 	Available as quickly as Skeleton Interrupt routine, if the mainline is being executed. Once execution is started, only interruptable by Skeleton Interrupt Routine or internal interrupt. Can be different with each mainline core load. Interrupt core load is also required if lower priority interrupts are serviced by interrupt core load. Must Call INTEX as last logical statement.
Interrupt Core Load Core Storage Location 	Large core area available. Once execution is started, only interruptable by Skeleton Interrupt Routine or internal interrupt. Mainline or nonprocess program in operation at time of interrupt is saved before and restored after Interrupt Core Load operation. Can Call DPART as last logical statement if interrupt is sometimes recorded; otherwise, should Call INTEX.
Mainline Core Load Core Storage Location 	Large core area available. Can include interrupt routines for other interrupts. Queued for execution if record indicator is on when named in QIFON statement. Must Call DPART for last logical statement if core load is sometimes executed at interrupt level; if always queued, should Call VIAQ.

Table 1. Types and Characteristics of Process Interrupt Servicing Routines

I/O operation is in progress unless its associated interrupt routine and I/O area are in the skeleton. This requires an exchange operation (an operation wherein a specified portion of the variable area of core is saved and the interrupt core load is read in for execution). Following execution of the interrupt core load, the original operating program is restored.

The option of recording or servicing any external interrupt may be different from one mainline core load to the next. The designation is made by control cards when the core load is being formed.

Programmed Interrupt - CALL LEVEL

This statement causes an interrupt (by programming) on any assignable interrupt level (0-23). The format is:

CALL LEVEL (I)

where

I is an integer constant (0-23) that specifies the interrupt level desired.

This call, which can be used only in process (mainline or interrupt) programs, causes a pseudo ILSW bit to be set on the level specified.

Programmed interrupts are treated the same as process interrupts in that they can be recorded or serviced, in-core or out-of-core, etc. The programmed interrupt servicing routines must follow the rules of process interrupt servicing routines. There can be only one programmed-interrupt routine per assignable interrupt level.

The programmed interrupt is recognized immediately when called from a lower level. When the servicing routine exits to MIC, program operation at the calling level is resumed with the statement following the CALL LEVEL statement.

A programmed interrupt called from a higher level is recognized after the calling program is completed and after any intervening interrupts are serviced. If a level is called and any ILSW bit is on when the level is recognized, the programmed interrupt is recognized after the first ILSW bit that is on is serviced.

Interrupt Exit-CALL INTEX

All interrupt routines serviced on an interrupt level must return control to MIC. The CALL INTEX statement, which has no parameters, is normally used for this purpose. It must be used as the last logical statement in skeleton interrupt routines, mainline interrupt routines, and it can be used in interrupt core loads.

Subprograms called by user-written interrupt servicing routines must use a RETURN statement to return to the interrupt routine, or a CALL DPART statement to return control directly to MIC. Figure 10 shows an example of the RETURN, CALL INTEX, and CALL DPART statements.

RECORDED INTERRUPT SERVICING

External interrupts whose occurrences are recorded are serviced with mainline core loads. The mainline core load performing the servicing is the same as any other mainline core load, except it is queued for execution by a CALL QIFON statement. Since it is a queued core load, it should have a CALL VIAQ as the last logical statement. (It could, of course, be the first core load of a special series and, as such, would end with CALL CHAIN to get the next core load of a sequence, but a CALL VIAQ eventually must be executed.)

COMBINATION CORE LOAD

In the descriptions given thus far there is only one major difference between an interrupt core load and a mainline core load used for servicing recorded interrupts. That difference is in the last logical statement, which must be CALL INTEX for an interrupt core load and CALL VIAQ for the mainline core load.

If an external interrupt is serviced immediately some times and recorded other times, it requires two core loads that might be the same in all respects, except for their last logical statement. To eliminate this situation, a combination exit statement, CALL DPART, is provided.

Departure-CALL DPART

The CALL DPART statement causes the level of operation to be tested and

1. If the present level is an interrupt level, a CALL INTEX is executed.
2. Otherwise, a CALL VIAQ is executed.

Thus, CALL DPART eliminates duplication of core loads. An interrupt that is sometimes recorded and sometimes serviced, when it occurs, can be serviced under either condition with the same core load. The core load operates from an interrupt level when servicing is specified; it is queued and operates from the mainline level when the interrupt is specified as recorded.

NOTE: The combination core load described above must not violate the restrictions placed on either mainline core loads or interrupt core loads. In other words, mainline interrupt subroutines are not allowed as part of this core load; only statements allowed in both mainline and interrupt programs are permitted (See Appendix B). Combination core loads are built in the same manner as a mainline core load (see Figure 4).

INTERRUPT ASSIGNMENT RESTRICTIONS

The following interrupt assignment restrictions must be observed for proper operation of the TSX system.

1. All I/O device interrupts must be assigned to a higher priority interrupt level than external interrupts unless the external interrupt is serviced by a skeleton interrupt routine.

The sequence of operations (specified by the encircled numbers) can be either 1, 2, 3, 4, 5, 6A, 6B, 6C, 8, 9, 10, or 1, 2, 3, 4, 5, 7A, 7B, 8, 9, 10.

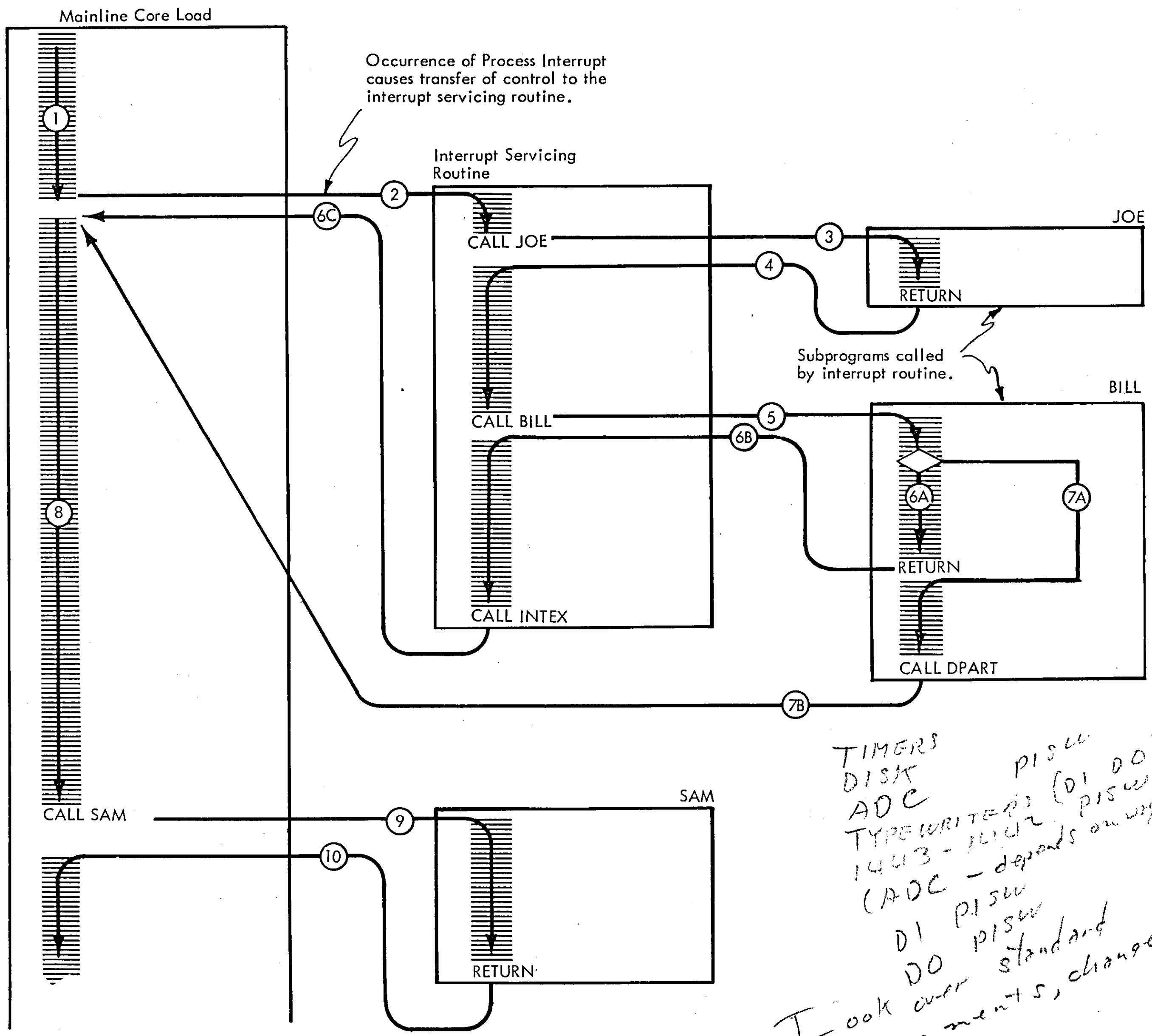

Mainline Core Load
Occurrence of Process Interrupt causes transfer of control to the interrupt servicing routine.
Interrupt Servicing Routine
1
2
3
4
5
6A
6B
6C
7A
7B
8
9
10
CALL JOE
CALL BILL
CALL INTEX
JOE
RETURN
Subprograms called by interrupt routine.
BILL
RETURN
CALL DPART
CALL SAM
SAM
RETURN

Figure 10. Examples of RETURN, CALL INTEX, and CALL DPART Statements

TIMERS
DISK PISW
ADC
TYPEWRITERS (DI DO) PISW
1443 - 1442 PISW
(ADC - depends on urgency)
DI PISW
DO PISW
Look over standard assignments, change

2. If external interrupts and I/O devices are both assigned to the same level, the external interrupts must be serviced by skeleton interrupt routines.
3. A skeleton interrupt routine cannot use an I/O device whose interrupt is assigned to the same or a lower priority level, except for disk, 1053 printer, and 1443 printer; however the 1053 test function cannot be used.

no disk swap until I/O is finished

4. ILSW bits must be assigned continuously beginning with position 0.

PISW ASSIGNMENT RESTRICTIONS

PISW (Process Interrupt Status Word) groups can be assigned to interrupt levels either as a single group per level or in multiple groups per level. The following rules and restrictions must be observed for proper operation of the TSX system.

One Group Per Level

Normal usage of process interrupts requires that only one group of process interrupts be assigned to each interrupt level. Process interrupts assigned in this way can each be serviced with separate interrupt routines. The servicing routines must reside in the skeleton area only if their associated interrupt level is the same as or higher than any I/O device interrupt level.

When only one PISW is connected to a level, the correlation of the interrupt level number to the PISW group number is as follows.

Interrupt Level	PISW Group
0	1
1	2
2	3
3	4
.	.
.	.
22	23
23	24

The MIC program does the ILSW and PISW sensing and transfers control to the proper interrupt servicing routine.

Multiple Groups Per Level

In special cases it is desirable to have more than one PISW group assigned to an interrupt level and this is possible with the TSX system; however, the following restrictions must be observed. The interrupt servicing routine must:

1. Reside in the skeleton area.
2. Sense all PISWs assigned to the level.
3. Upon completion, exit to MIC via the I/O exit (BSC I 90).

When assigned in this way there is no correlation restriction between the interrupt level number and PISW group number.

Combination PISW Assignments

It is possible to combine the two assignment methods and have some interrupt levels with only one PISW each and some levels with more than one PISW. The same rules and restrictions for each type as outlined above still apply. For example, to have two groups of four PISWs each assigned to interrupt levels 4 and 5 one valid combination is:

Interrupt Level	PISW Group
0	1
1	2
2	3
3	4
4	one group of four PISWs
5	one group of four PISWs
6	7
7	8
.	.
.	.
17	18
18	Not assignable; usage assumed on levels 4 and 5.
19	
.	
.	
23	

Any combination can be used for the PISW assignments on levels 4 and 5.

The user written routine used to service the interrupts must be coded as an I/O RPQ subroutine.

INTERVAL TIMER CONTROL PROGRAM

The interval timer control (ITC) program provides for FORTRAN language control of four types of timers:

1. Two machine interval timers (A and B).
2. Nine programmed interval timers.
3. A programmed real-time clock.
4. A timer for time-sharing control.

The ITC also performs three additional functions.

1. Resets the operation monitor during time sharing.
2. Tests for no response from 1053 printers.
3. Performs end of time sharing.

The third machine interval timer (C) is used for items 2, 3, and 4.

The time base periods available for the machine interval timers are .125, .25, .5, 1, 2, 4, 8, 16, 32, 64 and 128 milliseconds (ms). The .125-ms period is available only on a 2 μsec machine, and 128 ms is available only on a 4μsec machine. Each timer is assigned a permanent time base by the user. All three timers can operate at different time periods, or the same time period, but all three timers must be assigned to the same interrupt level. In order to schedule programs based on hours, minutes, or seconds, the wired-in base time for interval timer C must be an even divisor of one second (e.g., .5, 1, 2, 4, 8).

Machine Interval Timers

The two machine interval timers should be used to measure relatively short time intervals. They are controlled by the following statement.

```
     CALL TIMER (NAME, I, INT)
```

where

NAME is the name of the user's subprogram to be executed when the specified time elapses (NAME must also appear in an EXTERNAL statement; see the publication, IBM 1800 FORTRAN Language, Form C26-5905).

I is an integer expression, whose value must be:
1-for machine interval timer A. (word 4)
2-for machine interval timer B. (word 5)

INT is a positive integer expression that specifies the number of intervals counted before the user's subprogram is executed.

The subprogram specified in a CALL TIMER statement must be in core storage when the interrupt generated by the timer is recognized. The interrupt occurs when the time specified has elapsed, but it is recognized only when the level of operation is lower than the timer interrupt level and the timer level is unmasked. The timers are stopped and reset to zero when the specified time has elapsed and the interrupt is recognized (zero is a not-busy condition).

It is the user's responsibility to ensure that the subprogram NAME is in core when the timer interrupt is recognized. This can be accomplished in two ways:

1. NAME is a skeleton subroutine ~~resides in the skeleton~~.
2. NAME is a mainline subroutine, ~~interrupt routine~~, all interrupt levels with out-of-core interrupts are masked, and the core load exit is not allowed while the timer is busy (see Example 5 following).

The subprogram name is automatically loaded with the calling core load (unless previously loaded with the system skeleton). Also, the subprogram must return control to the ITC program (RETURN statement or assembler language equivalent). The program is executed at the interrupt level to which the interval timers are assigned and cannot be recorded.

It is not recommended that periodic programs (programs initiated by internal timers) be executed on the timer level. If this is allowed to happen, some timer interrupts may be missed during execution of lengthy programs. The CALL LEVEL statement (see Programmed Interrupt) is designed to handle this situation, and in this case, should be used to create an interrupt at a lower level of machine operation. The periodic program is then executed at the programmed interrupt level.

Example 1: Assume machine interval timer A is wired for the .125 ms time base.

```
     CALL TIMER (SCAN1, 1, 35)
```

When this statement is executed, ITC initializes timer A (sets it to -35) to count 35 intervals and return control to the statement following the CALL TIMER statement. When 35 intervals (i.e., 35 x .125 ms, or 4.375 ms) have elapsed, an interrupt occurs and control is transferred to the subprogram named SCAN1.

Example 2: Assume machine interval timer B is wired for the 2 ms time base.

```
     CALL TIMER (SEC1, 2, 500)
```

This statement causes the ITC program to initialize timer B (sets it to -500) to count 500 intervals and return control to the statement following the CALL TIMER statement. When 500 intervals (i.e., 500 x 2 ms, or 1 second) have elapsed, an interrupt occurs and control is transferred to the subprogram named SEC1.

Example 3: Assume machine interval timer A is wired for the 1 ms time base.

```
        .
        .
        .
  1 CALL TIMER (MIL5H, 1, 500)
        .
        .
        .
        .
        .
  2 CALL TIMER (MILL2, 1, 2)
        .
        .
```

Assuming the time (500 ms) specified in statement 1 has not elapsed, the timer A interrupt will occur 2 ms after execution of statement 2, and the sub-program named MILL2 will be executed. The sub-programmed named MIL5H will not be executed since timer A was reset before time elapsed. Although this condition can be prevented as shown in the following example, its logic is useful under certain conditions.

Example 4: Same conditions as for Example 3. This example shows the use of the LD subroutine in testing for a timer busy condition (see the section entitled Subroutines).

```
        .
        .
        .
  1 CALL TIMER (MIL5H, 1, 500)
        .
        .
        .
        .
        .
  2 IF (LD (4)) 2, 3, 3

  3 CALL TIMER (MILL2, 1, 2)
```

Statement 2 tests for timer A busy. If it is busy (negative in core location 00004) a programmed loop is performed until timer A is no longer busy (MIL5H execution completed), at which time statement 3 is executed.

Example 5: This example shows the use of the LD subroutine in testing for a timer busy condition. This test is required if SUBR7 is not in the skeleton and time-sharing is utilized.

```
        .
        .
  CALL TIMER (SUBR7, 1, 10)
        .
        .
        .
  12 IF (LD (4)) 12, 13, 13

  13 CALL VIAQ
```

In this example, statement 12 tests for timer A busy and waits until SUBR7 has been executed before going to the CALL VIAQ statement.

Clock and Programmed Timers

The programmed real-time clock and the nine programmed interval timers are updated by the third interval timer (C).

The time interval used for updating the clock (termed the interrupt time base) is the product of the wired-in time base interval and a number chosen by the user at system generation time. For example, assume interval timer C is wired for an 8-ms time base, and the clock is to be updated every second. The number necessary to accomplish this is 125 (8 ms x 125 = 1000 ms, or 1 second), and when the ITC program is assembled, the number -125 would be specified by the user. The third timer would then cause an interrupt every second. A minus number must be specified because the interval timer is incremented and causes an interrupt when it reaches zero.

Summary. Interrupt time base = wired-in time base x assigned number.

The interrupt time base is used specifically for the programmed real-time clock and as a primary base for the programmed timers and time-sharing clock. It is also used as the reset interval for the operation monitor during time-sharing operations.

Clock

The programmed real-time clock maintained by the interval timer control program records time to the nearest thousandths of an hour. Clock accuracy depends on the assigned interrupt time base previously described. The clock is reset on a 24-hour basis (i.e., it is incremented from 00:000 to 23:999 and then goes to 00:000).

To set the clock at a desired time, the following statement is used.

```
CALL SETCL (I)
```

where

I is an integer expression that specifies the desired time setting. The time setting must be expressed in hours and thousandths of hours (i.e., 00000 through 23999).

To read the programmed real-time clock, the following statement is used.

```
CALL CLOCK (I)
```

where

I is an integer variable where the time is to be stored.

Programmed Timers

The nine programmed timers should be used to specify long time periods. In particular, they can be used for periodic program execution or to initiate execution of a program at some later time.

If the called program is in the skeleton when the specified time elapses, the program is executed. The called program must return control to the ITC program (RETURN statement or assembler language equivalent). The program is executed on the same level that the interval timers are on, unless CALL LEVEL is used.

If the called program is not in the skeleton when the specified time elapses, it must be in the same form as a mainline core load. Out-of-core programs are handled as recorded interrupts, i.e., the program will not be placed in the queue until requested by a CALL QIFON statement, and will not be executed until a CALL VIAQ finds that the queued program is the highest priority in the queue.

To provide the user with large time intervals, ~~the CALL COUNT statement,~~ a larger time base can be specified for the programmed timers. The programmed timer base for the programmed timers is a user-assigned multiple of the interrupt time base used for the programmed real-time clock. For example, if the interrupt time base is one second, and the user wants the programmed timers to operate at 15 second intervals, then 15 is specified when the ITC program is assembled.

Summary. Programmed timer base = interrupt time base (previously assigned) x assigned number.

The programmed timer base is used specifically for the programmed timers and the time-sharing clock. This base is the smallest interval of time that can be specified for the programmed timers or for nonprocess program operation (i.e., time-sharing, see Time-Sharing Control Program).

The programmed interval timers are controlled by the following statement.

```
CALL COUNT (IN, I, INB)
```

where

IN is an integer constant or integer variable that specifies the number (0 through 31) of the program to be executed or recorded when the specified time elapses. These numbers are assigned by the user when the program core load is prepared.

I is an integer expression that specifies the number (1 through 9) of the programmed interval timer.

INB is an integer expression that specifies the number of intervals to be counted before the called program can be executed (multiple of programmed timer base). Program numbers are used in place of names to provide the recorded interrupt option.

Examples

Conditions:

    8 ms, wired time base for timer C
x 125, user-assigned number,
    1 second (1000 ms), interrupt time base,
x 15, user-assigned number,
    15 seconds, programmed timer base.

Statements:

```
    MINUT = 4

    IHOUR = 240

1   CALL COUNT (28, 1, 144)

2   CALL COUNT (28, 1, 36* MINUT)

3   CALL COUNT (31, 4, 1920)

4   CALL COUNT (3~~0~~1, 4, 8* IHOUR)

5   CALL COUNT (19, 6, 240)

6   CALL COUNT (19, 6, IHOUR)
```

Meanings:

Statement Numbers	Subprogram Number	Programmed Timer Number	Time Interval
1,2	28	1	36 minutes
3,4	31	4	8 hours
5,6	19	6	1 hour

The programmed timers can also be tested for busy status with a procedure similar to that used to test the machine timers.

Example 1.

Problem:

Do not execute subprogram 30 if subprograms 6 or 20 are in the process of being called.

Assume: Timer 1 is only set up to call subprogram 6; timer 4 is only set up to call subprogram 20 (see Subroutines).

Solution:

```
          .
          .
          .
    IF (LD (62)) 9, 2, 9
2   IF (LD (71)) 9, 3, 9
3   CALL COUNT (30, 8, 4)
9         .
          .
          .
```

Example 2. Variable periodic scan.

Problem:

Queue an analog scan program every five minutes with a priority of seven if J TEST (a programmed indicator) is set to zero; if non-zero, queue the same program every minute with a priority of one.

Assume:

1. Subroutine 19 is in the skeleton area; therefore, a busy test such as illustrated in example 1 is not required.
2. Timer interval and user-assigned expansions are the same as for previous example (8 ms x 125 x 15).

Solution:

The following call must be given in an initial core load.

```
    CALL COUNT (19, 4, 20)
```

This designates that subroutine 19 is to be called in five minutes. Thereafter, subroutine 19 calls itself within the specified time period.

```
    SUBROUTINE A        (Number 19 assigned at
                        load time)

    EXTERNAL SCAN

    IF (JTEST) 10, 20, 10 (Test Indicator)

10  IPER= 4             (Set up for period of 1
                        min. and priority of
    IPRI = 1            1.)

    GO TO 30

20  IPER = 20           (Set up for period of
                        5 min. and priority of
    IPRI= 7             7.)

30  CALL COUNT (19,     (Call this subr., 19 to
    4, IPER)            cause periodic execution.)

    CALL QUEUE          (Queue periodic scan core
    (SCAN, IPRI, 0)     load.)

    RETURN              (Return to ITC)

    END
```

SCAN is the name of a mainline core load that will be executed at mainline level as a result of a CALL VIAQ when SCAN is the highest priority entry in the queue.

In order to effect immediate execution of the scan routine, replace the CALL QUEUE statement with a CALL LEVEL statement to cause an interrupt on a lower level. The scan routine, in this case, is written as a skeleton interrupt servicing routine associated with the programmed interrupt on the level designated in the CALL LEVEL statement.

Operation Monitor

If the Operation Monitor is in use, the Time-Sharing Control program initiates action for the ITC program to automatically reset the Operation Monitor during time-sharing operations. This option can be omitted from the system during initial system assembly, in which case the user is responsible for maintaining the Operation Monitor at all times.

NOTE: The time interval selected for the Operation Monitor must be greater than the programmed timer base selected for the ITC program.

TIME-SHARING CONTROL PROGRAM

The time-sharing control (TSC) program controls the amount of time allowed for nonprocess program operations.

Time-sharing can be initiated in two ways:

1. Execution of a CALL SHARE statement in a process mainline program.
2. Execution of a CALL VIAQ statement when the core load queue table is empty. This causes the VIAQ subroutine to execute a CALL SHARE statement.

The first method can be utilized when time-sharing is desired at specific times and for different durations. When time-sharing is initiated in this way, the process core load is saved and the non-process monitor (or an unfinished program) is read into core and executed. When the specified amount of time has elapsed, the nonprocess program is saved (if not completed) and the process core load is restored. The maximum time for a time-sharing operation initiated in this manner is set by each CALL SHARE statement. Operation of the process core load is resumed with the statement following CALL SHARE.

The second method permits time-sharing when the computer is not being utilized for the process. The maximum time for a time-sharing operation initiated in this manner is specified in a control card by the user when the system is loaded, and remains constant. At the completion of the specified time, another CALL VIAQ is automatically executed by the system. If, in the interim, a core load has been queued, it is then executed; however, another time-sharing operation will be initiated if nothing has been entered into the queue.

Normally the CALL VIAQ method is used, but in special cases, the CALL SHARE method is also desirable.

All interrupts that occur during the time-sharing operation are handled by MIC the same as if a process mainline program were in operation. After the interrupt is serviced (or recorded) control is returned to the nonprocess program unless a CALL ENDTS statement is executed in the interrupt routine.

If the nonprocess program is not completed before time runs out, it is saved and continued when the next time-sharing operation is executed.

The following statement is used to initiate time-sharing operations for a specified time interval.

```
CALL SHARE (I)
```

where

I is an integer expression that specifies the number of time intervals allowed for nonprocess program operation.

The basic time interval is assigned by the user when the ITC program is assembled (see Interval Timer Control Program).

The CALL SHARE statement transfers control to the nonprocess monitor supervisor program or directly to the program that was operating when the last time-sharing operation was terminated.

Examples.

Assume that the programmed timer base is 15 seconds (the same as for the examples shown for the CALL COUNT statements).

Time-Sharing Interval Requested	Statement to Obtain Request
1 minute	CALL SHARE (4)
5 minutes	CALL SHARE (20)
30 seconds	CALL SHARE (2)
1.75 minutes	CALL SHARE (7)

The time-sharing operation is terminated whenever the time interval specified by the user has elapsed, and it is usually not terminated before. That is, if the user specifies one minute for time-sharing, it is usually one minute before control is returned to the statement following the CALL SHARE statement.

The nonprocess monitor performs a WAIT operation if there are no jobs to be performed. Interrupts are still serviced as they occur, and if an interrupt routine recognizes a need for the process program to resume operation, it can terminate the time-sharing operation by executing the following call:

CALL ENDTS

This call can be used only in an interrupt routine, and it sets the time sharing clock to indicate zero time. The first timer C interrupt that occurs after control is returned to the nonprocess program causes the time-sharing operation to be terminated and control is then returned to the process mainline program.

If time-sharing is not in effect, the CALL ENDTS statement has no effect.

Two additional functions performed by TSC are CALL LINK and CALL EXIT when called from non-process programs.

ERROR ALERT CONTROL PROGRAM

The Error Alert Control (EAC) program receives control from

1. Any input/output subroutine when the subroutine cannot correct an error or interrupt condition
2. The queue subroutine when the core load queue table would overflow
3. The master interrupt control program when an internal machine error occurs (i.e., invalid operation code, parity, or storage protect violation)
4. Other control programs

Upon entry (EAC is entered at word 120), EAC receives the error identification and other pertinent data. From this information, the core and disk portions of EAC will perform the following operations:

1. Optionally, dump core storage to disk (not performed for internal machine errors).
2. a. If in a nonprocess program terminate the program if the error cannot be operator corrected.
   b. If in a process program branch to the user-written error subroutine that is with the core load (this step is bypassed for internal machine errors or if an error subroutine is not included).
3. Update error counters maintained on disk.
4. Execute a subroutine (IBM written) for the device or error condition, print an error message on the EAC printers, and set up possible recovery action.
5. a. If in a nonprocess program return control to the monitor supervisor program if recovery is not possible.
   b. If in an interrupt routine terminate the program, service any other interrupts, and perform the action specified by the user or the device error subroutine.
   c. If in a mainline core load perform the action specified by the user or the device error subroutine.

When the EAC program is initially assembled, the option of the core dump in item 1 can be selected.

Also, when assembling the skeleton programs, a back-up unit of the same type can be specified for 1053, 1816, and 1443 printers. Backup for the EAC printer is achieved by defining multiple EAC printers at TASK assembly time (if the EAC printer is defined as a 1053). When an output error occurs, or if the unit is not ready (interrupt response not received), EAC will logically disconnect the unit in error and substitute the back-up unit. When backup is initiated because of a hardware malfunction, the message in progress on the failing unit is not continued on the backup device. When the error condition is corrected, the unit can be restored to its original status by using the C. E. interrupt routine. See C. E. Interrupt Routine in the publication IBM 1800 Time-Sharing Executive System Operator's Guide, Form C26-3754.

Error Subroutine

A user-written error subroutine can be optionally included with each process core load. The purpose of this subroutine is to allow the user to have control before EAC overlays the variable area with the disk portion of EAC. For example, there may be special data or other information that the user wants to save. Output, such as special core dumps, messages, or contact operate functions, can also be executed. The error subroutine cannot be written in FORTRAN language.

Before entering the user's error subroutine, error identification data is placed in words 00115-00119. The words will contain the following:

Input/Output Errors.

00115	Error type code
00116	Address of illegal call or address of the device table
00117	Address of level work area
00118	Address of originating call

Queue Overflow.

00115	Error type code
00116	Word count of core load named in CALL QUEUE
00117	Sector address of core load named in CALL QUEUE
00118	Priority of core load named in CALL QUEUE
00119	Error parameter of core load named in CALL QUEUE

The error type codes are listed in Table 2.

A standard recovery procedure is executed by EAC according to the type of error (see EAC EXIT column, Table 2). User options are specified in the same table (see USER OPTN column). However, under certain conditions, EAC overrides the user option. The EAC option is always executed if an error subroutine is not used or the user does not specify an option. Options can be specified by the user before returning to EAC by loading the A-register with -10 for S (RESTART) or -1 for I & R (CONTINUE).

The last logical statement in the error subroutine must be BSC I Entry to the error subroutine.

The core load named for the restart option can be an error analysis core load, or it can be the first of a new series of core loads. If queuing techniques are used, the restart core load can be simply a CALL VIAQ statement (CALL QUEUE can be executed in the restart core load or the error subroutine).

The statements listed below cannot be used in an error subroutine.

```
CALL BACK        CALL QIFON
CALL CHAIN
CALL DPART       CALL RESMK
CALL ENDTS       CALL SAVMK
CALL EXIT        CALL SHARE
CALL INTEX       CALL SPECL
CALL LEVEL       CALL UNMK
CALL LINK        CALL VIAQ
CALL MASK
```

LOADERS

Prior to execution of any user written programs, the TSX system must be loaded on the disk and the system skeleton must be built and loaded. The programs provided for these functions are:

TASK
System loader
Skeleton builder
Core load builder

Temporary Assembled Skeleton (TASK)

The TASK program is an IBM supplied program that initially controls the system loader, skeleton builder, and core load builder. When the system skeleton is loaded to core, it overlays TASK.

System Loader

The prime functions of the system loader are to load the TSX system programs, build an interrupt assignment table from user-supplied control records, and prepare the disk for system operation.

Control records are required to supply the system loader with information that relates to the interrupt level assignment of I/O units, interval timers, and process interrupts.

From the information supplied, the system loader makes entries in the location equivalence table (LET) for

1. Disk communication area
2. Master branch table
3. Skeleton subroutine entry table
4. Supervisor program
5. Disk utility program (DUP)
6. Assembler
7. FORTRAN
8. Simulator
9. Location equivalence table (LET)
10. Subroutine library
11. Nonprocess work storage area
12. Nonprocess program save area
13. Process program save area
14. Message buffer area
15. Core image program area
16. Interrupt save area
17. Process work storage area
18. Special mainline save area
19. System skeleton
20. Error alert program save area
21. Error alert program dump area
22. Error decision programs
23. Process cold start program

Skeleton Builder

The skeleton builder program obtains information from user-assigned control records, programs and

DEVICE/ROUTINE	ERROR CODE	EAC EXIT	USER OPTION	ERROR DESCRIPTION
1053/1816 PRINTER/KEYBOARD	00	S	N	ILLEGAL CALL
	01	R,S	S*	1053 NOT READY
	02			NOT USED
	03	R,S	R,S	1816 KEYBOARD NOT READY
	04	L	N	STORAGE PROTECT VIOLATION FROM 1816
	05	S	R	KEYBOARD PARITY ERROR
	06	I	N*	PRINTER PARITY ERROR
	07	R	N*	NO PRINT RESPONSE
	08	R	N	IN VALID MESSAGE ON DISK
	09			NOT USED
1442 CARD READ-PUNCH	10	S	N	ILLEGAL CALL TO 1442
	11			NOT USED
	12	R	S	PARITY ERROR
	13	L	N	STORAGE PROTECT VIOLATION
	14	R	S	FEED CHECK AT READ STATION
	15			NOT USED
	16	R	S	READ-PUNCH CHECK, DATA OVERRUN, OTHER FEED CHECKS
	17	S	N	//BLANK CARD
	19	R	S	1442 NOT READY
1054/1055 PAPER TAPE READER/PUNCH	20	S	N	ILLEGAL CALL
	21	S	I	PUNCH PARITY ERROR
	22	R,S	S	READER NOT READY
	23	R,S	S	PUNCH NOT READY
	24	S	I	READER PARITY ERROR
	25	L	N	READER STORAGE PROTECT
	26-29			NOT USED
2310 DISK	30	S	N	ILLEGAL CALL
	31	R	S	DISK NOT READY
	32	S	I	DATA OVERRUN
	33	S	I	WRITE SELECT
	34	S	I	DATA ERROR
	35	L	N	STORAGE PROTECT ERROR
	36	S	I	PARITY ERROR
	37	S	N	INVALID DISK ADDRESS
	38	S	N	FILE PROTECT ERROR
	39			NO RESPONSE
1627 PLOTTER	40			NOT USED
	41	S	I	PARITY ERROR
	42	R,S	S	NOT READY
	43-49			NOT USED
1443 PRINTER	50	S	N	ILLEGAL CALL
	51-52			NOT USED
	53	R,S	R,S	NO PRINT RESPONSE
	54	S,I	I*	PARITY ERROR
	55	R,S	R,S*	NOT READY
	56-59			NOT USED
ANALOG INPUT BASIC	60	S	N	ILLEGAL CALL
	61	L	N	STORAGE PROTECT VIOLATION
	62	S	N	PARITY CONTROL ERROR
	63	S	N	PARITY DATA ERROR
	64	S	N	OVERLAP CONFLICT
	65			INTERMEDIATE INTERRUPT
	66-68			NOT USED
	69	I	N	COMPARATOR VIOLATION

● Table 2. EAC Error Type Codes (Part 1)

DEVICE/ROUTINE	ERROR CODE	EAC EXIT	USER OPTION	ERROR DESCRIPTION
DIGITAL INPUT BASIC	70	S	N	ILLEGAL CALL
	71	S	N	PARITY FRROR
	72	L	N	STORAGF PROTECT ERROR
	73			INTFRMEDIATE INTERRUPT
	74-79			NOT USED
DIGITAL AND ANALOG OUTPUT BASIC	80	S	N	ILLEGAL CALL
	81	S	N	PARITY FRROR
	82			INTFRMFDIATE INTFRRUPT
	83-89			NOT USED
2402 MAGNETIC TAPE	90	S	N	ILLEGAL CALL
	91			NOT USED
	92	L	N	STORAGE PROTECT VIOLATION
	93	S	R	COMMAND REJECT
	94	S	R	EXCESSIVE TAPE ERRORS
	95	R	S	TAPE ERROR
	96-97			NOT USED
	98	R, S	S	NOT READY
	99	R	S	END OF TAPE
FORTRAN	100	S	N	ILLEGAL ADDR COMPUTED IN AN INDEXED STORE
	101	S	N	ILLEGAL INTEGER USED IN A COMPUTED GO TO
DISK I/O	102	S	N	FILE NOT DEFINED
	103	S	N	RECORD TOO LARGE, ZERO OR NEGATIVE
NON-DISK I/O	104	S	N	INPUT RECORD IN ERROR
	105	S	N	RANGE OF NUMERICAL VALUES IS IN ERROR
	106	R	N	OUTPUT FIELD TOO SMALL TO CONTAIN NUMBER
	107	S	N	ILLEGAL UNIT REFERENCE
	108	S	N	REQUESTED RECORD EXCEEDS ALLOCATED BUFFER
	109	S	N	WORKING STORAGE AREA INSUFFICIENT FOR DEFINE FILES
MISCELLANEOUS	110	S	N	PSC CALL BACK ERROR
	111	S	N	CORE LOAD NOT LOADED ON DISK
	112	L	N	RESTART CORE LOAD NOT LOADED ON DISK
	120	S	N	ERROR OPTION IS ZERO CALL IGNORED
				ERROR OPTION NOT ZERO NO LOWER PRIORITY IN QUEUE
				QUEUE ENTRY IS REPLACED BY NEW CALL
	130	S	N	CALL RESMK ERROR
	131	S	N	CALL UNMK ERROR
	140	S	N	INTERRUPT LEVEL ERROR
INTERNAL ERRORS	996	S, L	N	CAR CHECK ERROR
	997	S, L	N	OP CODE VIOLATION
	998	S, L	N	STORAGE PROTECT VIOLATION
	999	S, L	N	PARITY ERROR

LEGEND:

I -	CONTINUE AT THE POINT OF INTERRUPT
R -	RETURN TO THE ROUTINE WHICH DETECTED THE ERROR
S -	RESTART
L -	RELOAD
N -	NO OPTION - MUST TAKE EAC EXIT
* -	INTERNAL BACKUP ATTEMPTED

● Table 2. EAC Error Type Codes (Part 2) subroutines, and information provided by the system loader to construct the system skeleton (in core-image form) and then store it on disk. The skeleton can be read into core, for execution, by a cold start operation.

The rebuilding of the system skeleton is required any time routines are added or deleted, or other modifications are made.

Core Load Builder

The core load builder is a program that converts user-written programs and subroutines into core loads (process mainline, interrupt, or nonprocess) for storage in the core image area on disk. All process core loads must be built and stored on disk prior to execution under control of the TSX system. The control records supplied by the user provide the core load builder with the names of the relocatable mainline, interrupts to be recorded, data files to be used, interrupt routines to be included as part of the core load, and LOCAL subprograms.

Using the information provided by the system loader and the skeleton builder, as well as the information from the programs and subroutines, the core load builder generates the tables, transfer vectors, and work areas that are combined with the instructions that make up a core load.

NONPROCESS MONITOR

The nonprocess monitor provides the user with a programming tool that simplifies the task of generating, organizing, and testing programs executed under control of the 1800 TSX System and to supervise execution of nonprocess programs. The nonprocess monitor can be operated on-line under control of the system director, or it can be operated off-line. Off-line operations are possible after system generation. Initial assembly of certain system programs and user's beginning routines are done under TASK control.

The primary function of the nonprocess monitor is to provide continuous processor-controller operation during a sequence of jobs that might otherwise involve several independent programming systems. The monitor coordinates the processor-controller activity by establishing a common communications area in core storage, which is used by the various programs that make up the monitor. It also guides the transfer of control between monitor programs and the user's programs. Operation is continuous and setup time is reduced to a minimum, thereby effecting a substantial time saving in processor-controller operation and allowing greater programming flexibility.

Control records, which are used to direct the sequence of jobs without operator intervention, must be prepared prior to the actual operation. The control records are included in a stacked input arrangement.

The nonprocess monitor (Figure 11) is composed of five programs.

Supervisor Program. This program supervises all nonprocess monitor operations. It decodes the monitor control records in the stacked input for nonprocess jobs and calls the proper monitor program to perform the desired operation. For example, a typical sequence of jobs might be execution of a payroll program, compilation of a FORTRAN process control program, and simulation of a core load that includes the program just compiled. The supervisor program calls the program loader, FORTRAN compiler, and simulator program, respectively, to handle these jobs.

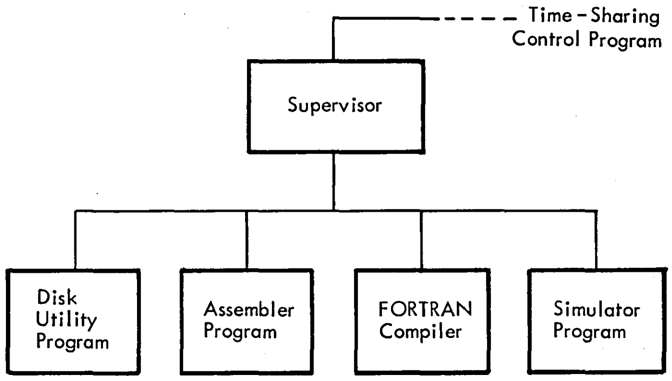


Figure 11. Nonprocess Monitor

Disk Utility Program (DUP). DUP is a set of routines designed to aid the user in performing the functions of disk maintenance. That is, it is capable of storing, deleting, and outputting user's programs and defining system and machine parameters. DUP updates the location equivalence table (LET) when a change is necessary and also maintains other communications areas.

FORTRAN Compiler. The compiler translates programs written in the FORTRAN language into machine language and automatically provides for calling the necessary arithmetic, functional, conversion, and input/output subroutines.

Assembler Program. The assembler translates programs written in symbolic language into machine language. Basically, it is a one-for-one type assembly program. That is, the assembler usually produces one machine language instruction for each symbolic instruction of the source program. Provision is also included for the user to easily make use of input/output, conversion, and arithmetic subroutines that are a part of the subroutine library.

Simulator Program. This program is designed to provide a means for simulating a process control or data acquisition program without interfering with the normal operation of the process. Each input/output call sequence of the program being simulated is analyzed and simulated individually.

Various options are offered throughout the simulator. For example, an analog input call sequence can obtain input from three sources: cards, paper tape, and a random number generator. The option chosen is specified in a control record read by the simulator program.

SUPERVISOR PROGRAM

The supervisor program performs the control functions and directs the loading operations for the nonprocess monitor. The FORTRAN compiler, assembler program, disk utility program, simulator program, and the program loader are called for operation under control of the supervisor.

The supervisor program operates under control of the system director, and can be called for operation by the CALL SHARE statement in a process mainline program (or the nonprocess monitor can be operated off-line).

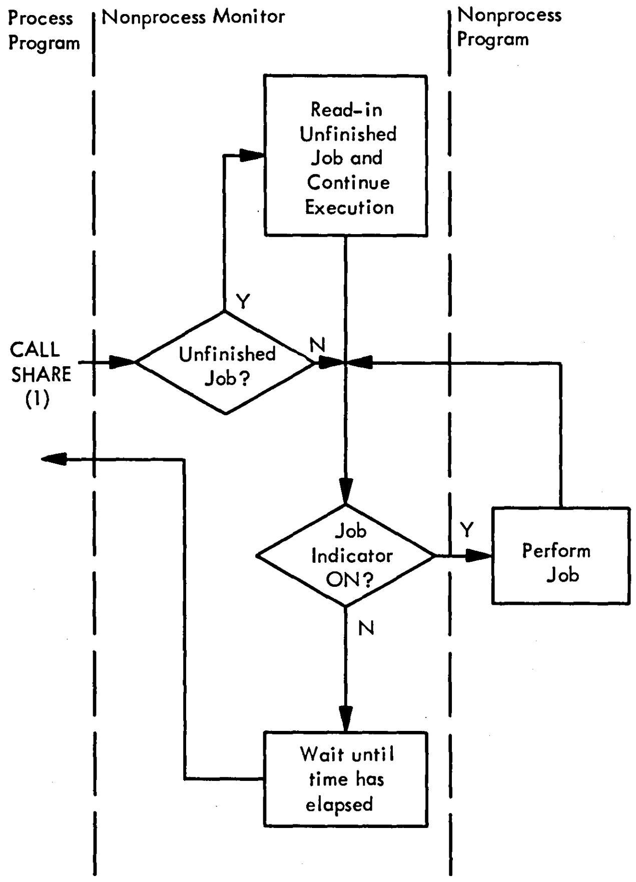
During on-line operation of the nonprocess monitor, process interrupts are serviced as they occur, though the interrupt servicing time is counted against the time allotted for nonprocess operations. For example, assume a process program calls for one minute of time-sharing. If, during the one minute period, ten seconds are used to service process interrupts, only 50 seconds are actually available for the nonprocess job.

If all nonprocess jobs are completed before the end of the specified time, the supervisor program performs a Wait operation for the remainder of the time. In other words, if the CALL SHARE statement specified one minute of time-sharing, control is not returned to the process program until one minute has elapsed unless a CALL ENDTS statement is executed by an interrupt routine (see Time-Sharing Control Program).

If a nonprocess job is not completed before the specified time is up, it is saved on the disk. When the next CALL SHARE statement is executed, operation of the nonprocess job is resumed where it left off.

When an unfinished job is waiting, the CALL SHARE statement causes it to be read in and executed. Otherwise, the supervisor program is read into core and determines by checking a program indicator located within the system director if any time-sharing operations are to be performed. This indicator is turned on by the execution of a special console interrupt routine, supplied with the system.

When nonprocess jobs are ready to be executed, the operator places the input in the card or paper tape reader, sets the required configuration of the program switches, and presses the console interrupt key. The interrupt routine called for execution is the one that sets the indicator to its ON status. The indicator can be turned off only by an END OF ALL JOBS record in the stacked input.



SUPERVISOR COMPONENTS

The supervisor program is a group of several distinct, but closely related routines. The basic components are

- Skeleton supervisor
- Monitor control record analyzer

SKELETON SUPERVISOR

The skeleton supervisor is read into core storage when the operation of the monitor system is initially started. The skeleton supervisor, which always resides in core during monitor operation, provides the communications link between the monitor programs and the user's programs; i.e., the skeleton supervisor contains the necessary logic to conduct the transition from one program to another.

MONITOR CONTROL RECORD ANALYZER

The monitor control record analyzer routine analyzes the monitor control records, prints out the information contained in the control record, and calls the appropriate program.

Monitor Control Records

The nonprocess monitor control records can be entered from cards only. All control records start with two slashes and a blank. The control identification starts in the fourth position. A list of the control records and their operations follows.

JOB: The JOB record causes termination of a previous job and initialization for a new job (see Job Deck). This includes identification of the packs to be used with the job and resetting initial status of indicators. When a job is aborted, as when an uncorrectable I/O error occurs, records in the card reader are passed until the next JOB record is recognized.

END OF ALL JOBS: This record must be the last item in the stacked input. It is used to indicate that there are no more nonprocess operations to be performed. To restart the nonprocess monitor operation, the console interrupt routine must be executed.

ASM: The ASM record causes the assembler to be read into core storage for execution. The assembler control records and source statements for the program to be assembled must follow directly behind the ASM monitor control record. These control records are used to cause a listing to be printed, an object program to be punched, the system symbol table to be used with the assembly, and to specify other options. The name of a mainline program that is assembled is punched in the ASM record. If the assembly does not contain any errors, the object program will be loaded as a temporary program, with a temporary entry of the name in the LET. Such entries are retained for the duration of the job in progress.

FOR: The FOR record causes the FORTRAN compiler to be read into core storage for execution. The FORTRAN control records and source statements for the program to be compiled must follow directly behind the FOR monitor control record. These control records are used to specify precision for real variables, integer word size, trace options. listing options, I/O units to be used by the mainline and subprograms, and other options. The name of the mainline program is punched in the FOR record. All successfully compiled programs are temporarily stored, with temporary entries in the LET. Such entries are retained for the duration of the job in progress.

DUP: The DUP record causes the disk utility program to be read into core storage for execution. The disk utility program requires control records for direction (e.g., STORE, DUMP, DELET), and DUP records must follow directly behind the DUP monitor control record. The DUP records are explained in the disk utility program section.

XEQ: The XEQ record causes a user's program to be read into core storage for execution. The program may be read in from either the core load area or the relocatable program area on disk. The required subroutines are loaded with the main program. If specified, a core map is printed during the loading of a core load from relocatable programs. This provides a listing of the beginning core storage addresses of the mainline program and subroutines. The LOCAL, FILES, INCLD, and CCEND control records can be used when building a core load for execution; however, these control records cannot be used if the XEQ record specifies a program in the core load area. These records are described under the DUP STORECI heading.

SIM: The SIM record causes the simulator to be called for simulation of a nonprocess core load. Process core loads must be simulated using the SIMULCI control record. The format of the SIM record is identical to that of the XEQ.

PAUS: The PAUS record causes a WAIT instruction to be executed to allow the operator to make setup changes. The monitor operation proceeds when the console START key is pressed. An interrupt that occurs during the wait period is serviced but control is returned to the WAIT instruction.

*(COLUMN 4): The asterisk is used to identify a comments record. The information in this record is printed on the LIST and SYSTEM printing units.

Job Deck

The input to the nonprocess monitor consists of one or more job decks. Each job deck is preceded by a JOB control record. The last job must be followed by a JOB and END OF ALL JOBS records. The processing of each job deck is controlled by the supervisor program as specified in the monitor control records. As an example, consider the following stacked input arrangement. (The sequence of operation is from top to bottom.)

```
// JOB
// FOR A
.
. (Source program A)
.
// DUP
*STORE A
// FOR B
.
. (Source program B)
.
// XEQ A
*CCEND
// XEQ B     only non process
*CCEND
// JOB
// END OF ALL JOBS
```

The sequences listed above compile, store, and execute both program A and program B, providing, (1) there are no source program errors, and (2) there is sufficient room in the work storage area. A source program error found in a FORTRAN compilation or an assembly causes the DUP store operation to be bypassed for that program. Dump operations, if specified by *PUNCH control records, are performed and all following XEQ requests preceding the next JOB record are disregarded.

This feature (XEQ – request disregarded) can prove very useful when successful execution of one program depends upon the succesful completion of the previous program. A combination such as this should be considered as one job and the XEQ control records should not be separated by a JOB record.

DISK UTILITY PROGRAM

The disk utility program (DUP is a group of generalized routines that assist the user in day-to-day operation of his installation. By means of these routines, the most frequently required operations of disk maintenance can be performed with a minimum effort.

The disk utility program is called for operation by a DUP monitor control record.

Control Statements

The DUP record must be followed by at least one control statement that selects the routine desired. The DUP control statements are identified by an asterisk in column one. Columns two through ten contain a code word to identify the routine (e.g., STORE, DEFINE). The columns after the code word provide additional information to be used by the routine itself.

*STORE The store routine is used to store relocatable programs in the relocatable program area on disk. The program can be loaded from cards or from the temporary program area. The drive where the program is to be stored can be specified. Program names are assigned at the time this routine is used. The name specified for a program must not already be in the LET, except that the same name can reference two different programs if one is a temporary program.

*STOREDATA The store data routine is used to store a block of data on the disk and to assign a name to it. Data can be stored from cards or the nonprocess work storage area. Each data block assigned a name is stored in the first available sectors of the fixed area on the drive specified. If the data block is stored in the nonprocess work storage area, it is not assigned a name and it is stored in the first sectors of the area.

*STOREMOD The store modify routine moves a program or data from the user area over a program. nonprocess core load, or data block specified by name, providing the old area will hold the modified version. Thus, it can be used to replace an object program already stored on disk.

*STORECI The store core image routine is used to store a program in core image form (i.e., as a core load) in the core load area and to assign the core load a name. A map of the locations and the names of subroutines and subprograms loaded in the core load area can be obtained. The relocatable mainline can come from cards or from the disk drive specified. The drive where the core load is to be stored can also be specified. Four types of core loads are possible; nonprocess core loads, mainline core loads, interrupt core loads, or combination interrupt and mainline core loads. These are specified by a blank, an M, an I, or a C in the control record. If a mainline core load is specified, the user must include the name of a restart core load in the record. If an interrupt core load is specified, the level and the PISW bit position must be included. Both a restart and a level and bit position must be specified when a combination core load is loaded. The name of the core load as well as the name of the relocatable main program must be specified in all cases. When a mainline core load is correctly built, DUP will make a search of the program name table that is with every other mainline and interrupt core load. When it finds the name of the core load, that was just built, it adds the disk address and the word count for the core load to the name table. Also, any programs named in the name table of the new core load are looked up in the fixed location equivalence table (FLET) and the disk addresses and word count of these programs are added to the table. Any core load names of programs that have not yet been loaded are ignored. The disk addresses will be added when the missing core load is stored at some time later. The same operations take place with nonprocess core loads that call other core loads (CALL LINK). Only nonprocess core loads are examined when a nonprocess core load is built.

The interrupt core loads have addresses stored in ICLT (interrupt core load table), which is maintained in the skeleton. This table is updated by using the level and bit indicators to determine where the new core load sector address and word count should be inserted.

The combination core load will have its disk address stored in ICLT, to satisfy the interrupt core load requirement. Of course all the core loads that have the combination core load name in their name tables will have the disk address added to the name tables.

Five different control records are used to indicate special core load requirements. They are the RCORD, INCLD, FILES, LOCAL, and CCEND records. These control words are punched in columns two through six and an asterisk is placed in column one. The FILES, LOCAL, INCLD, and CCEND can be used after an XEQ control record, after a SIM control record, after a STOREMD control record, or after a STORECI control record for a nonprocess core load. These are used to build nonprocess core loads. Process core loads (loaded with STORECI or SIMULCI) permit all five records.

*RCORD control records are used to specify the level and PISW bit positions for interrupts that are to be recorded if they occur during the execution of the core load. Only mainline core loads and combination core loads need this control record.

*INCLD control records are used to specify the interrupt subroutines by name, level, and bit that are included with the mainline or combination core load; to specify the trace and error subroutine to be used with the mainline, interrupt, combination, or nonprocess core load; or to specify the programmed interrupt programs by name and level that are to be included with the mainline or combination core loads.

*FILES control records are used to establish an equivalence between a symbolic file number used in a FORTRAN DEFINE FILE statement and the name in FLET of a data area or, the disk drive for the data area when it is to be at the end of work storage. These records can be used to build any type of core load previously described.

*LOCAL control records are used to identify what subprograms are to be loaded into core only when they are called by the programs that are permanently in core with the core load. It is possible to read in more than just the program being called by specifying several programs in one LOCAL block. Once in core, the block will remain in the area reserved for LOCALs until it is overlayed by another LOCAL. No program that is already in the LOCAL area will be overlayed unless a program that is a LOCAL not currently in core is called.

The *CCEND control record is used to signal that no more loader control records follow; however, it cannot be used with an XEQ record that specifies a program in the core load area. Other than the CCEND control record, all loader control records can be multiples.

*DICLE The define interrupt core load entry control record is used to cause the same interrupt core load, which must already be on the disk, to be used to service more than one interrupt. All process interrupts on any level can call in the same core load, and the particular interrupts and interrupt core load name are specified with this control record. When an interrupt that has no other service routine occurs, as during the initial use of the system, it is advisable to have an error core load that will be executed if the interrupts do occur. The same core load can be used to handle all such spurious interrupts.

*SIMULCI The simulate core image routine is used to store a process program in core image format for simulation. The program is placed in nonprocess work storage for the simulator. SIMULCI has the same format as STORECI and serves a similar purpose for a simulation run. If the program being simulated may require additional simulation, it is advisable that the user precede the SIMULCI card with a STORE card (the location in working storage of the program being simulated is changed when the next // JOB card is read).

*DUMP This routine is used to dump (unload) programs from the disk to one of the following: cards, list printer, or from a program area to a nonprocess work storage area.

*DUMPDATA This routine is used to dump data or a core image program from disk to one of the following: cards, list printer, or from a program area to a nonprocess work storage area.

*DUMPLET This routine dumps the contents of the location evuivalence table (LET and/or FLET) to the list printer. The first line will contain the LET (or FLET) header words and the following lines will contain the alphabetic name and the disk address. The drive to be used can be specified as well as the part (fixed or relocatable) of the table to be output.

*DELET This routine is used to delete a named program, core load, or data file from the disk. A mainline, combination, or interrupt core load that is called from any other core load or as the result of an interrupt will not be deleted unless a replacement core load, named in the DELET card, is available. A link that does not have a replacement named in the DELET control record will be deleted but the word count in the program name table of any other core loads that could call the deleted link will have a negative value. This will cause an invalid disk address error message to be printed if the link that has been deleted is called for.

*DEFINE The define routine is used to define variable parameters required by the system. Items that can be defined include the number of disk drives, size of core storage, the packing of programs on disk, the specific system programs to be removed from disk, and the location and size of the disk areas used by the system. When redefining areas for optimum placement or change in the amount of disk required, the DEFINE routine permits a great amount of flexibility. For example, the areas for special save interrupt save, FORTRAN I/O save, 1053 message buffering, and dumps to disk can be selectively omitted. Also, a core load area, a process work storage, a nonprocess work storage area, and an area for the storage of relocatable programs can be optionally included on any or all disk drives. This routine will separate file protected cylinders from nonfile protected cylinders when the user places two such areas next to one another, or it will pack the same type of unprotected file areas or protected file areas when the user makes it possible by the placement of his areas. The nonprocess work storage, the message buffer, the error save, interrupt save, and error dump areas are not file protected. The user can control the relative placement of his areas by the sequence in which he specifies these areas. To begin with all newly labeled disk packs contain only a label, a LET, and nonprocess work storage area. When an area is defined to be on such a pack, the area it requires is subtracted from the high addressed cylinders of nonprocess work storage on that pack. Another area defined on that pack will occupy the next lower area below the area previously defined to be on that pack, etc. The LET is maintained as the definitions are executed.

One important restriction that exists is that no core image programs or data areas can be removed from the core image area as a result of a definition operation. For that reason it is important to define the areas as soon as possible after the system is obtained. Later re-definitions are possible, but some additional problems can occur.

The DEFINE routine _must_ be executed before the skeleton is built. The operation establishes the size of the skeleton in words and the size of the largest interrupt core load area in words. Both of these areas should be established with the possibility that the requirements for these areas can expand. Thus, they should almost certainly be established with extra words to more easily expand them at a later time.

New packs can be defined with core load area and with process work storage area while the system is on-line with the process. Definition of all other areas to different drives should only be done while the system is not on-line.

Deleting of relocatable programs does not cause other programs to be packed over the area of the deleted program. At convenient intervals, the user should repack the disk using the DEFINE PAKDK control record. The drive containing the pack to be packed is specified. When this operation is performed the user should be certain he has a current record of disk storage in the event that a solid disk error should occur while the packing is in progress. The amount of time that the operation requires will increase with the amount of data that must be moved in order to secure a packed disk.

Any of three different system programs can be removed from the system using the DEFINE REMOV control record. These are the simulator, FORTRAN, and the assembler. When one of these is removed, the area it occupied is incorporated into nonprocess work storage.

*SEQCH This routine is used to change the sequence in which mainline core loads are executed. This is accomplished by modifying the program name tables of specified core loads that call a given core load so that a PSC call (CHAIN, SPECL, QUEUE, or UNQ) to a given core load now refers to the core load designated by the modified sequence. Up to eight core loads can be specified for a change in sequence with one SEQCH control record.

*DLABL The define label routine is used to place an identifying number in the first sector of the pack. When a non-system drive is specified, DLABL will cause a LET area, containing entries that define the balance of the pack as available area, to be created starting in the second sector of the pack. Sector addresses will be written on the pack.

*DWRAD This routine is used to write addresses within a specified area on a disk pack. The user can zero the data area or retain the previous data, and initiate or remove file protection. During these operations, the addresses can be verified with those already on the disk. This function can be used to unprotect those areas in the core load area that contain FORTRAN files that are to be written as well as read by the FORTRAN programs.

ASSEMBLER PROGRAM

The assembler program for the 1800 TSX system is a disk resident version of the 1800 card assembler. The language for the assembler is described in the publication, IBM 1800 Assembler Language (Form C26-5882).

General Operation

The ASM monitor control record is used to call the assembler program into operation. The assembler reads the control records and the source program in card form only. After assembly, if no errors were detected by the assembler, the object program resides in the user's program area as a temporary program and can be called for execution by an XEQ control record, or it can be stored permanently in the user area by a DUP operation.

Assembler Control Records

The assembler control records are used to control the assembler operation and must be in card form. They must precede all source statements of the program.

An asterisk in column one precedes the control record name.

TWO PASS MODE This control record causes the object program to be produced through two passes of the source program. The TWO PASS MODE record is required when the space allotted for the nonprocess work storage area to be used by the assembler is too small to contain the intermediate output during assembly.

LIST DECK This record causes uncompressed object output to be punched in cards. Source statement errors are indicated by codes punched in columns 18 and 19.

The LIST DECK record requires the TWO PASS MODE record because the source program must be read twice. During the second pass, uncompressed object information is punched into the card just read.

LIST DECK E This record is identical to LIST DECK, except that the uncompressed object information consists only of the error columns (18 and 19). If the source statement contains no errors, columns 18 and 19 are left blank.

LIST This record causes a printed listing to be generated on the List printer during the second phase of the assembly. The List printer can be a 1053, 1816, or 1443 Printer.

PUNCH This control record causes the compressed object output to be punched. This output will be punched, even if errors are discovered in the source input by the assembler.

PUNCH SYMBOL TABLE This record causes the symbol table to be punched upon completion of the assembly operation.

PRINT SYMBOL TABLE This record causes the symbol table to be printed on the List printer upon completion of the assembly operation.

SYSTEM SYMBOL TABLE This record causes the system symbol table to become a part of the symbol table to be used in the assembly. Thus, the symbols in the system symbol table can be used without defining them in the source program.

SAVE SYMBOL TABLE This record causes the symbol table used in the assembly to be saved in the system symbol table area of disk storage. Symbols previously stored are overlaid by the new symbols.

OVERFLOW SECTORSbn This record indicates the number of sectors (n) of nonprocess working storage the assembler allows for possible symbol table overflow. The range is 1-32.

COMMONbn This record allows a COMMON area to be saved for linking from a FORTRAN mainline program to an assembly mainline and then back to the FORTRAN mainline. n is the length of COMMON in words.

FORTRAN COMPILER

The TSX FORTRAN compiler is a disk resident version of the 1800 card compiler. The FORTRAN language is described in the publication, IBM 1800 FORTRAN Language (Form C26-5905).

Operation

The FOR monitor control record is used to name the FORTRAN mainline program and call the FORTRAN compiler into operation. The compiler reads the control records and source program in card form only. After compilation, the object program resides in the user's program area as a temporary program, and can be called for execution by an XEQ control record, or it can be stored permanently in the user's program area by a DUP operation. All FORTRAN programs are compiled in relocatable format.

FORTRAN Control Records

The following control records are used to control the compilation operation.

*IOCS (CARD, MAGNETIC TAPE, TYPEWRITER, KEYBOARD, 1443 PRINTER, PAPER TAPE, DISK, PLOTTER) This record must be used to specify any I/O device required for execution of the program; however, only the devices required should be included. Because the *IOCS card can appear only in the mainline program, it must include all the I/O devices used by all FORTRAN subprograms that will be called. The device name must be in parentheses with a comma between each name.

*LIST SOURCE PROGRAM The source program is listed as it is read in.

*LIST SUBPROGRAM NAMES The names of all subprograms (including EXTERNAL subprograms) called directly by the compiled program are listed.

*LIST SYMBOL TABLE The following are listed:

- Variable names and their relative addresses
- Statement numbers and their relative addresses
- Statement function names and their relative addresses
- Constants and their relative addresses
- Unreferenced statement numbers

*LIST ALL The source program, subprogram names, and symbol table are listed. If this record is used, the other *LIST control cards are not required.

*TRANSFER TRACE The compiler generates linkage to trace routines whenever an IF statement or computed GO TO statement is encountered. If Data Entry switch 15 is on at execution time, the trace printout routine prints the value of the IF expression or the value of the Computed GO TO index (see Optional Tracing).

If tracing is requested, an *IOCS control card must also be present to indicate that either the 1816/1053 Printer or the 1443 Printer is needed. If both the 1816/1053 Printer and the 1443 Printer are indicated in the *IOCS card, the 1443 Printer is used for tracing.

The traced value for the assignment of a variable on the left of an equal sign of an arithmetic statement is printed with one leading asterisk. For the expression of an IF statement, the traced value is printed with two leading asterisks. The traced value for the index of a Computed GO TO statement is printed with three leading asterisks.

*ARITHMETIC TRACE This control record will cause trace instructions for arithmetic statements to be compiled with the object program. Data Entry switch 15 must be on (the same as for TRANSFER TRACE above) at execution time in order to effect the trace printout. Also, an *IOCS control record must be present to indicate the printer type (see Optional Tracing).

*EXTENDED PRECISION This record compiles 3-word real constants and real variables to provide extended precision for arithmetic operation.

*ONE WORD INTEGERS This control record causes the integer variables in a nonprocess program to be allocated one word of storage in the object program, instead of two words for standard precision or three words for extended precision. All integer variables in a process program are automatically made one word.

*NONPROCESS PROGRAM This control record is required for all nonprocess programs to differentiate them from process programs. The presence of this record prevents the usage of certain TSX CALL statements as defined in Appendix B. The absence of this record forces *ONE WORD INTEGERS.

**(HEADER INFORMATION) If the 1443 Printer is the List printer, the information contained in card columns 3-72 is printed at the top of each page printed out during the compilation.

OPTIONAL TRACING

The user can elect to trace only selected parts of the program by placing statements in the source program logic flow to start and stop tracing. This is done by executing a CALL to either subroutine:

```
CALL TSTOP (to stop tracing)
CALL TSTRT (to start tracing)
```

Thus, tracing occurs only if:

- The trace control records were compiled with the source program.
- Data Entry switch 15 is on (can be turned off at any time).
- A CALL TSTOP has not been executed, or a CALL TSTRT has been executed since the last CALL TSTOP.

1053/1816 CONTROL

The 1053 and 1816 output printers can be controlled by outputting a unique value for the operation desired. The value must be assigned as an integer constant and outputted through A-conversion.

The operations that can be performed and the unique values assigned to them are

Backspace	5696
Carrier Return	5440
Line Feed	9536
Shift to print black	5184
Shift to print red	13632
Space	16448
Tabulate	1344

As an example of the printer controls, assume that a variable, X, is to be printed in the existing black ribbon shift and that another variable, Y, is to be printed in red following a tabulation. Following the printing of Y, the ribbon is to be shifted back to black. This can be accomplished as follows:

```
I =   1344
J =  13632
K =   5184
WRITE (1,3) X, I, J, Y, K
FORMAT (F12.6, 2A1, F12.6, A1)
```

Unit 1, as specified in the WRITE statement, is assumed to be a 1053 or 1816 printer. The sequence of operations to be performed are: print the variable X, tabulate, shift to print red, print the variable Y, shift to print black.

Each control variable counts as one character and must be included in the count of the maximum line length.

Programming Note

As a general rule, the user should use a slash as the final format specification in an interrupt program. This allows the printer to take a line space prior to returning to a possible print operation that was interrupted during mainline execution. The extra line space is required to prevent possible overprinting which can occur as a result of an interaction between the mainline and interrupt programs or between two interrupt programs.

SIMULATOR PROGRAM

The simulator program allows the user to check process and nonprocess core loads for possible programming errors. Since it is designed as an aid in debugging process programs without disrupting the normal operation of the TSX system, the simulator can be operated on a time-sharing basis without removing the 1800 System from its normal on-line status.

Each instruction in the object program is analyzed for a valid operation code and valid format before its operation is simulated. Also, the addresses of store instructions are checked to ensure that the instruction will not change anything outside of the areas of COMMON, the defined program, INSKEL COMMON, or the work level area.

The system skeleton routines are used during simulation, and therefore, the skeleton area must be built before simulation can be performed.

Since the primary design function of the simulator is to detect programming errors in the object program that, if executed, would be detrimental to the process under control, the following built-in debugging features are provided to aid the user.

SNAPSHOT. This feature provides a display of the registers* after an instruction, whose location has been specified on the *SNAP control card, has been executed.

BRANCH TRACE. This feature provides a display of the registers* after a branch or skip has been executed. The location limits of the branch or skip are specified on the *TRACE control card.

*These are psuedo-machine registers maintained by the simulator.

DUMP. This feature dumps simulated core within the limits specified by the *DUMP control card.

The simulator operates under control of the nonprocess monitor. Simulation runs for process programs are called by a DUP control record, *SIMULCI. Simulation runs for nonprocess programs are called by a // SIM monitor control card. The operating procedures and stacked input for a simulation run are described in the publication, IBM 1800 Time-Sharing Executive System Operating Procedures (Form C26-3754).

SUBROUTINES

General Input/Output

Each time the simulator encounters a user-called sequence to an I/O subroutine, the location of the calling sequence and the subroutine name are printed on the list printer. Each time the simulator encounters a subroutine test function (I/O function digit = 0), the following occurs: the first time a test is encountered, a busy return is made; the second time, a not busy return is made. Succeeding entries alternately cause busy and not busy returns.

Listed below are the general input/output subroutines (IBM-supplied) recognized by the simulator, and the corresponding operations which the simulator performs:

SUBROUTINE	OPERATION
CARDN (Simulated card subroutine)	Read a card, feed a card, simulate punch a card
DISKN (Simulated disk subroutine)	Read disk, write disk, simulate disk seek
MAGT (Simulated magnetic tape subroutine)	Simulates all read, write, and control functions relative to 2401 and 2402 magnetic tape units
PAPTN (Simulated paper tape subroutine)	Simulate reading paper tape, simulate punching paper tape
PLOTX (Simulated plotter subroutine)	Simulate plotter output
PRNTN (Simulated printer subroutine)	Print a line, simulate a carriage operation
TYPEN or WRYTN (Simulated printer-keyboard subroutine)	Simulates printing on 1816 printer keyboard or 1053 printer

The simulator requires that the card reader, disk, and list printer be physically present on the system.

Process Input/Output

Call sequences which specify input from pulse input points, digital input points, process contact points, and analog input, may obtain the input from two sources: cards and a random number generator.

Data cards are used if samples of specific values are desired; the points can be read in a nonprocess program and punched into cards to be read by the simulator. Any value can be simulated when using cards, but in order to obtain the desired results, the input data must be sequenced according to the flow of the process input subroutines called. In other words, the card feature requires careful ordering of the card deck.

A random number generator, within the simulator program, produces numbers within a user-specified range. With this option, the user can employ a wide variety of input data to check the program operation. A psuedo-process input environment can also be created through the use of a random number generator.

All input values are printed on the list printer as they are called.

In the program being simulated, call sequences that specify output for the contact operate, pulse output, digital output, and digital-to-analog output features are printed when they are encountered. Input call sequences, error messages, and data are included in the printed output. This provides a complete chronological record of all that occurred during the simulation.

IBM-supplied process input/output subroutines are functionally simulated; that is, the subroutines' call parameters are analyzed according to specifications supplied in the form of control records. The routine name, calling parameters, and data are printed on the list printer. Listed below are the process input/output subroutines recognized by the simulator, and the corresponding operations which the simulator performs. Special condition returns are also simulated.

SUBROUTINE	OPERATION
AIPTN or AIPN (Simulated analog input point)	Simulates the read of a single analog point
AIRN (Simulated analog input random)	Simulates reading random analog input points
AISQN or AISN (Simulated analog input sequential)	Simulates reading sequential analog data points
DAOP (Simulated digital-analog output)	Simulates the transfer of digital or analog information
DICMP (Simulated digital input: read compare)	Simulates the reading in of digital input values under program control and compares these values to a table of user-supplied values. Only the first compare error is detected. A single entry to the special routine is made with appropriate indication. The end-of-table interrupt will not occur if a comparator error occurs.
DIEXP (Simulated digital input: read expand function)	Simulates the reading in of a digital input value and expands it into 1, 2, 4, 8, or 16 words.
DINP (Simulated digital input: hardware functions)	Simulates the reading in of digital input values

Arithmetic and Conversion Subroutines

Copies of the IBM-supplied arithmetic and conversion subroutines are contained within the simulator. It is these copies that are executed when a call to an arithmetic or conversion subroutine is encountered. The requested operations are performed in a psuedo-processing environment maintained under control of the simulator.

General TSX Subroutines

When a call to a TSX control subroutine is recognized by the simulator, the subroutine name and its calling sequence parameters are printed. There are two categories of subroutines designed for control and communication with the TSX system: the termination class and the functional simulate class.

The following subroutines comprise the termination class, and when encountered, cause the simulator to execute the close job procedure:

BACK	PAUSE
CHAIN	SPECL
DPART	STOP
INTEX	VIAQ

The subroutines listed below comprise the functional simulate class, and when encountered, cause the simulator to simulate the function, i.e., analyze the call parameters for validity and print the routine name, the calling parameters, and the data contained within the subroutine.

CLEAR	REMSK
COUNT	SAVMK
ENDTS	SETCL
LEVEL	SHARE
MASK	TIMER
OPMON	UNMK
QIFON	UNQ
QUEUE	

User-Written Subroutines

User-written subroutines are simulated in the same manner as the mainline program.

COMMON AREA

The simulated COMMON area can be dumped on cards whenever a program being simulated is terminated. The output cards can be used for input to reload COMMON, thus providing communication from one program to another.

RESTRICTIONS

Restrictions placed upon the use of the simulator program are listed below.

1. Nonprocess work storage must be used if actual data is to be transferred between disk and core.
2. Link or chain jobs must be simulated by presenting one core load at a time.
3. The simulator utilized LIBF and CALL instructions for special purposes. When analyzing post mortem dumps, the contents of LIBF and CALL locations should be ignored by the user.
4. All I/O must be performed by simulator subroutines. An execute I/O (XIO) instruction is not simulated but will be recorded on the list printer.
5. A wait (WAIT) instruction will be recorded on the list printer.
6. A storage protect setting instruction (STS with both the F-bit and the 9th bit equal to zero) will result in a termination.
7. An attempt to store into a skeleton area other than the INSKEL COMMON and work level areas will result in a termination.
8. Operation codes of 00, 38, 58, 78, and FF are invalid and will result in a termination.
9. A subroutine I/O area parameter pointing to the skeleton will result in a termination.

It is often necessary to repeat a group, or block, of instructions many times during the execution of a program (examples include conversions of decimal values to equivalent binary values, computation of square roots, and reading data from a card reader). It is not necessary to write the instructions each time a function is required. Instead, the block of instructions is written once, and the main program transfers to that block each time it is required. Such a block of instructions is called a subroutine.

These subroutines normally perform basic functions so that they may be used in the solution of many types of problems. When a main program uses a subroutine several times, which is the common situation, the block of instructions constituting the subroutine need appear only once. Control is transferred from a main program to the subroutine by a set of instructions known as a calling sequence, or basic linkage. A calling sequence transfers control to a subroutine and, through parameters, gives the subroutine any control information required.

The parameters of a calling sequence vary with the type of subroutine called. An input/output subroutine requires several parameters to identify an input/output device, storage area, amount of data to be transferred, etc.; whereas an arithmetic or functional subroutine usually requires one parameter representing an argument. The calling sequences used with the TSX system subroutines consist of a CALL or LIBF statement, whichever is required for a particular subroutine, followed by DC statements that make up the parameter list. The calling sequences for the various subroutines are presented later in this section.

The subroutines provided with the TSX System consist of the following groups:

1. Arithmetic and functional
2. Input/output and conversion
3. Selective dump and miscellaneous

The Arithmetic and Functional subroutines are those most frequently utilized for mathematical computations. The Input/Output (I/O) subroutines provide a quick and easy method for the programmer to reference the various I/O devices for input or output of data. The conversion subroutines provide a method of converting data from one code to another.

The selective dump subroutines allow the user to print selected areas of core storage during execution of an object program. The miscellaneous subroutines provide the user with the ability to perform certain machine operations using the FORTRAN language. Any of the subroutines in the TSX system can be assigned to permanent core. They then reside in the skeleton area of core storage and can be called by any program. Two of the I/O subroutines, DISKN and TYPEN or PRNTN) must be in the Skeleton area because they are used by the system director.

The user should keep in mind that subroutines are also placed in the skeleton area if they are called by skeleton interrupt routines. If the subroutine is used only occasionally or is of a large size, the user should consider placing it in variable core in order to better utilize core storage.

Programs that are executed from the variable section of core can also call the subroutines that are in the skeleton area. In fact, when core loads are being formed, the only subroutines that are placed with the core load are those that are not already in the skeleton area.

In order to allow entry from various programs and various interrupt levels before completing the computations from a previous call, the arithmetic, functional, and most of the I/O subroutines are designed as re-entrant routines; that is, they are re-entrant because they can be entered from a different level of machine operation despite the fact they may not have completed operation on a previous level.

Within the skeleton area there is a subroutine work area for each interrupt level being utilized. Thus, if one of the subroutines is in operation for the mainline program and an interrupt occurs, the same subroutine can be utilized by the interrupt servicing routine without destroying the partially completed mainline work.

NOTE: The only restriction that applies to the use of a subroutine is that the I/O subroutines (with the exception of the disk and printer subroutines) cannot be called from an interrupt level that is equal to or higher than the interrupt level associated with the device being used. (The disk and printer routines, which are described later, can be called from any interrupt level; see Printer Subroutine, or Disk Subroutine.

I/O SUBROUTINES

The TSX I/O subroutines were designed to reduce the amount of time spent by the programmer in accomplishing the input and output of data from and to the various input/output devices attached to the computer. They handle all of the details peculiar to each device, including the usually complex interrupt functions, and are capable of controlling many input/output devices simultaneously and asynchronously. In assuming the burden of the details of I/O operation, the subroutines permit the programmer's attention to be directed to the problem-solving aspects of each individual job, rather than accessory I/O housekeeping.

In order to better understand the subsequent descriptions of the individual I/O subroutines, the user should be familiar with certain characteristics which are common to the I/O subroutines, namely:

Methods of data transfer
I/O Subroutine operation
General error handling procedures
Basic calling sequence

METHODS OF DATA TRANSFER

IBM 1800 I/O devices and their related subroutines are grouped according to their method of transmitting and/or receiving data. There are two basic groups, direct program control and data channel.

Direct Program Control

Direct program control requires a programmed I/O operation for each word or character transferred. A character interrupt occurs whenever a character I/O operation is completed. This method is used for single-word analog or digital I/O and for the following devices: 1054/1055 Paper Tape Attachment, 1816 Printer-Keyboard, 1053 Printer, and 1627 Plotter.

Data Channel

A data channel requires an I/O operation only to initiate the data transfer. The device is provided with control information, word-counts, and data from the user's I/O area. Once initiated, the transfer takes place completely asynchronous to program execution. An operation-complete interrupt signals the end of the I/O operation when all of the data has been transmitted. The data channel is used for multiple-word transfers of analog and digital I/O data and for the following devices: 2401 and 2402 Magnetic Tape Units, 2310 Disk Storage, 1443 Printer, and 1442 Card Read Punch.

I/O SUBROUTINE OPERATION

This section briefly describes the internal makeup of the I/O subroutines. This description, along with some basic flow charts, will make it easier for the reader to understand the individual subroutine descriptions presented later in the manual.

Each I/O subroutine is divided into two routines: a call routine and an interrupt response routine. The call routine is entered when a user's calling sequence is executed; the interrupt response routine is entered as a result of an I/O interrupt.

Each subroutine saves and restores the contents of the A- and Q-registers, index registers, and the overflow and carry indicators except DISKN does not save the A-register and the process I/O subroutines (AIPIN, etc.) do not save the overflow and carry indicators. Also, most subroutines assume that the necessary interrupt levels are unmasked so that the associated I/O interrupts can be recognized (the disk and printer subroutines can operate from any interrupt level, masked or unmasked).

Call Routine

The call routine illustrated in Figure 12 has four basic functions:

1. Determine if any previous operations on the specified device are still in process.
2. Check the calling sequence for legality.
3. Save the calling sequence.
4. Initiate the requested I/O operation.

The flow diagram (Figure 12) is not exact for any one subroutine. It is only a general picture of the internal operation of a call routine.

Determine Previous Operation. The TSX I/O subroutines contain the extra instructions necessary to handle all of the devices of that type simultaneously. The WRTYN subroutine is an example. When an operation is started on one 1053, that device becomes busy. However, a subsequent call to the WRTYN subroutine specifying a different 1053 will be honored. A call to another I/O subroutine, such as PRNTN or CARDN is not affected by the fact that one or more of the 1053's is busy.

Check Legality of Calling Sequence. Calling sequences are checked for such items as illegal function character, illegal device identification code, zero or negative word count, etc.

Save Calling Sequence. The call routine saves within itself all of the calling sequence information needed to perform an I/O operation. The user may modify a calling sequence even though an I/O operation is not yet complete. However, the I/O data area should be left intact during an operation because the subroutine is continually accessing and modifying that area.

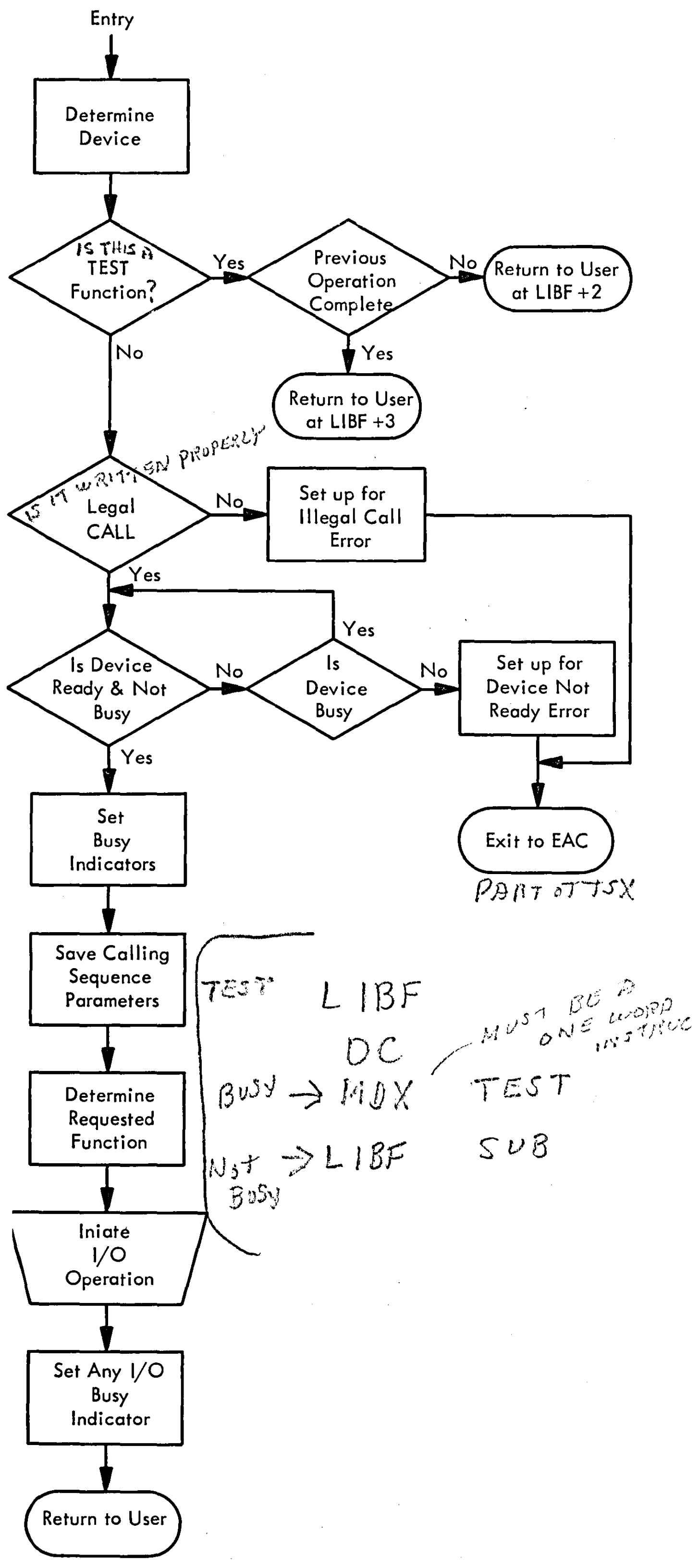

Entry
Determine Device
IS THIS A TEST Function?
Yes
Previous Operation Complete
No
Return to User at LIBF +2
Yes
Return to User at LIBF +3
No
IS IT WRITTEN PROPERLY
Legal CALL
No
Set up for Illegal Call Error
Yes
Yes
Is Device Ready & Not Busy
No
Is Device Busy
No
Set up for Device Not Ready Error
Yes
Set Busy Indicators
Exit to EAC
PART OT TSX
Save Calling Sequence Parameters
TEST LIBF
DC
BUSY → MDX TEST
NOT BUSY → LIBF SUB
MUST BE A ONE WORD INSTRUC
Determine Requested Function
Iniate I/O Operation
Set Any I/O Busy Indicator
Return to User

Figure 12. Call Routine

Initiate I/O Operation. The call routine only initiates an I/O operation. Subsequent character interrupts or operation complete interrupts are handled by the interrupt response routine.

Interrupt Response Routine

The I/O interrupt response routine, illustrated in Figure 13, is entered as a result of an I/O interrupt. The interrupt causes the user's program to exit to the MIC routine which in turn exits to the I/O interrupt response routine. The interrupt response routine checks for errors, does any necessary data manipulation, initiates character operations, and initiates retry operations in case of errors. It then returns control to MIC which returns control to the user.

Character interrupts for devices under direct program control occur whenever data may be read or written; e.g., a paper tape character read. Operation complete interrupts for all devices on the data channel (and some devices under direct program control) occur whenever the specified block of data has been read or written, e.g., a line printed or a card read.

Error detection and recovery procedures are an important part of a subroutine. However, little can be done about reinitiating an operation until a character interrupt or operation complete interrupt occurs. Therefore, the error indicators are not examined until one of these interrupts occurs.

A recoverable device is one which can be easily repositioned by the subroutine or operator and the I/O operation reinitiated. If the device is not recoverable or if the error cannot be corrected after a specified number of retries, the user is informed of the error condition.

GENERAL ERROR HANDLING PROCEDURES

Each subroutine has its own error detection routines. (In this context, the term "error" includes such conditions as last card, channel 9, end of tape, etc.) These routines categorize the error and choose an error procedure. Errors can be divided into two categories: those that are detected before an I/O operation is initiated, and those that are detected after an I/O operation has been initiated. Table 2 contains a list of the errors detected by the subroutines.

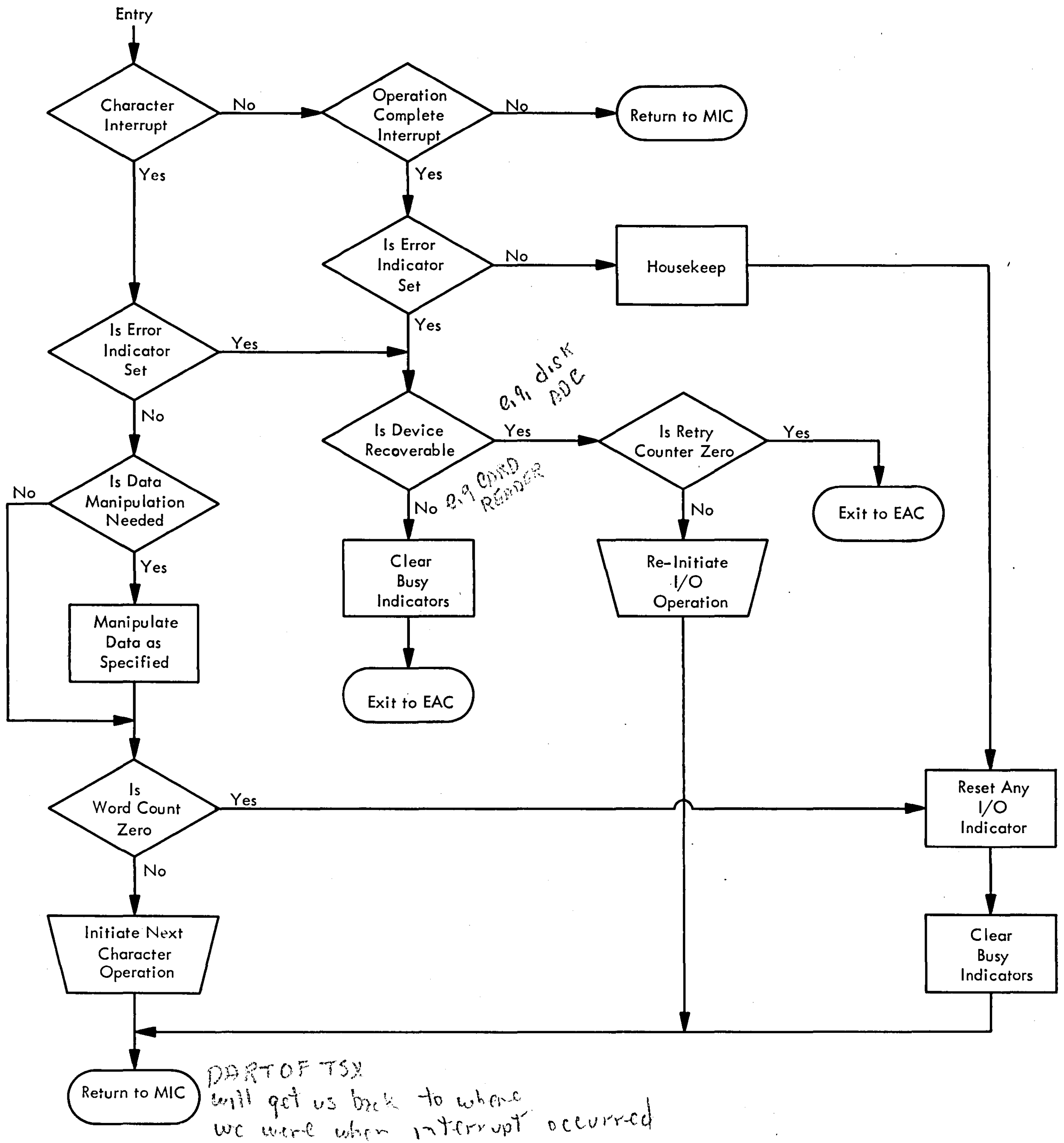


Figure 13. Interrupt Response Routine

Pre-Operation Checks

Before a subroutine initiates an I/O operation, it checks the status of the device and the legality of the calling sequence parameters. If the device is not ready or a parameter is in error, the subroutine exits to EAC.

Post Operation Checks

After an I/O operation has been started, certain conditions may be detected, about which the user should be informed. These conditions may be card jams for which manual intervention is needed in order to continue, read checks which have not been corrected after a specified number of retries, or indications of equipment readiness, such as end of tape marks. All of these conditions are detected during execution of the I/O interrupt response routine.

BASIC CALLING SEQUENCE

Each of the subroutines is entered via a calling sequence. These calling sequences follow a basic pattern; in fact, some are identical except for the name of the subroutine being called. In order not to burden the reader with redundant descriptions, this section presents the basic calling sequence and describes those parameters which are common to most of the subroutines.

Basic Calling Sequence

```
LIBF        Name
DC          Control parameter
DC          I/O area
DC          Special condition parameter
```

The above calling sequence, with the parameters shown, is basic to most of the TSX I/O subroutines. Unless otherwise specified, the subroutine returns control to the instruction immediately following the last parameter.

Name Parameter

Each subroutine has a symbolic name, which must be written in the LIBF statement exactly as listed in Table 3 because the core load builder must recognize the name to generate the proper linkage.

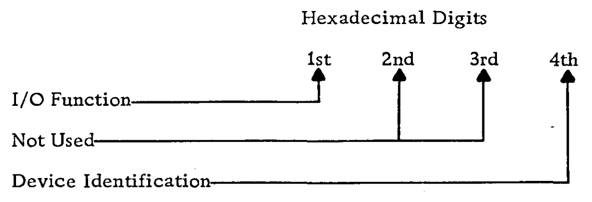
● Table 3. I/O Subroutine Names

Subroutine	Name
CARD	CARDN
Disk	DISKN
1443 Printer	PRNTN
Printer-Keyboard	TYPEN
1053 Printer	WRTYN
Magnetic Tape	MAGT
Paper Tape	PAPTN
Plotter	PLOTX
Digital Input	DINP, DICMP, DIEXP
Digital/Analog Output	DAOP
Analog Input	
Sequential Read	AISQN, AISN
Single Read	AIPTN, AIPN
Random Read	AIRN

Control Parameter

The control parameter, in the form of four hexadecimal digits, conveys necessary control data to the I/O subroutines by specifying the desired function (read, write, etc.), the device identification, and other similar control information. Most subroutines do not have use for all four digits.

A typical control parameter is illustrated below.



Since the I/O function and device identification are used in most subroutines, a description of the purpose of each is given here.

I/O Function

Each device is capable of performing a set of functions. The function digit in the calling sequence specifies which I/O operation the user is requesting. Four of these functions, read, write, test, and reset are used in most subroutines.

Read. The read function causes a specified amount of data to be read from an input device and placed in a specified input area. Depending upon the device, an interrupt signals the subroutine either when the next character is ready or when all requested data has been read. When the specified number of characters has been read, the subroutine becomes available for another call to that device.

Write. The write function causes a specified amount of data from the user's output area to be written (or punched) on an output device. As with the read function, an interrupt signals the subroutine when the device can accept another character, or when all characters have been written. When the specified number of characters has been written, the subroutine becomes available for another call to that device.

Test. The test function causes a check to be made as to the status of a previous operation on the same device. If the previous operation has been completed, the subroutine branches to the LIBF+3 core location; if the previous operation has not been completed, the subroutine branches to the LIBF+2 core location. The test function is illustrated below:

	LIBF	Name
LIBF+1	DC	Control Parameter (specifying Test function)
LIBF+2	OP Code	XXXX....
LIBF+3	OP Code	XXXX....

NOTE: Specifying the test function requires two statements (one LIBF and one DC), except in the disk subroutine where three statements are required.

This function is useful in situations where input data has been requested, and no processing can be done until that data is available.

The test function must also be used if the user assigns multiple I/O devices to one data channel (see Note following). If two or more devices are attached to the same channel, it is the user's responsibility to ensure that two devices are not operated simultaneously. In other words, if devices A and B are on the same data channel, a busy test for A or B must precede any function request for either device A or B. This is a user responsibility.

NOTE: The following require a data channel/device:
1. IBM 2310 Disk Storage
2. IBM 1443 Printer, if defined as error, list, or system printer
3. IBM 1442 Card Reader Punch
4. Analog input if continuous scan is utilized
5. Analog comparator if continuous scan with comparator is utilized.

Reset. The reset function terminates the I/O operation in progress.

Device Identification

In cases where it is possible to have more than one device of the same type on a system the user must specify in the calling sequence the device addressed.

The device identification digit must be 0 (zero) for the first device, 1 for the second device, etc.

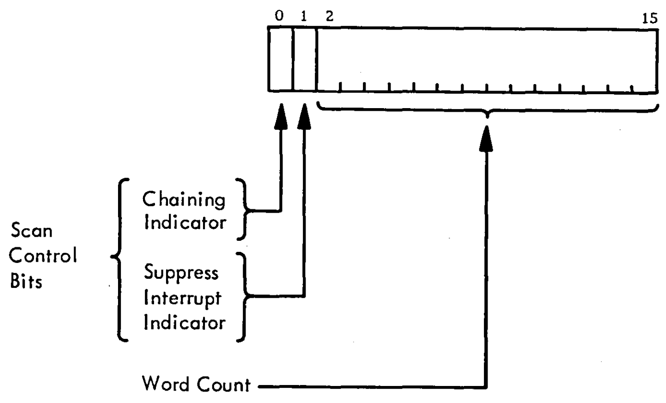
NOTE: In the Disk subroutine calling sequence, the device identification digit appears in the sector address word rather than in the control parameter.

I/O Area Parameter

The I/O area for a particular operation consists of one or more tables of control information and data. Each table is composed of a data area preceded by a control word (two control words for disk operations and some digital/analog operations) that specifies how much data is to be transferred. The area parameter in the calling sequence is the address (symbolic or actual) of the first control word that precedes the data area.

The majority of the subroutines allow one table per calling sequence. However, the magnetic tape and several digital/analog subroutines operating on separate Data channels permit tables to be "chained" together. This is accomplished through scan control bits in the control word preceding the data area of a table, and by following one table with the address of the next table.

The format of the control word used for all subroutines is shown below. The disk subroutine and certain analog/digital subroutines require a second control word which is described along with those subroutines.



The chaining bit, when set to 1, indicates that the associated table is part of a chain and that the word following the table is a chain-address word.

The suppress interrupt indicator bit, when set to 0, causes an interrupt after the data related to that table has been transferred. The resultant interrupt causes the TSX I/O subroutine to branch to the address specified in the error parameter, except when the table is the last table in the chain. In this case, the interrupt is used to initiate I/O subroutine housekeeping functions.

The word count refers to the number of data words in the table. It is important to remember that the number of words in the table is not always the number of characters to be read (or written) because some codes pack several characters per word.

Special Condition Parameter

This parameter is used only for the AIRN, AISQN, DICMP, MAGT, DINP, and DAOP routines and is described with them.

For compatibility reasons it must be included in most I/O subroutine calling sequences even though, in most cases, it is a pseudo parameter and may be defined as any constant.

CARD SUBROUTINE

The card subroutine performs all I/O functions relative to the IBM 1442 Card Read-Punch; read, punch, feed, and select stacker.

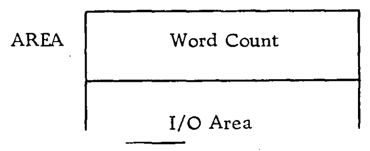
The card subroutine is not re-entrant for a given device, though two different 1442's may be called from different levels. CARND may be located in the skeleton I/O area or within a core load.

If CARDN determines a 1442 is not ready on an I/O operation, it will exit back to the user after notifying EAC. The I/O operation that was requested by the user will be retained by CARDN and will be executed the next time it is called, before any new I/O operation, if the 1442 is ready. If the 1442 is not ready, CARDN will wait until it is ready before executing both calls. Therefore, the user should give a test function after each I/O function, so that, if the 1442 was not ready at the time of the call for the I/O function, the function can still be executed when it does become ready.

When the Last Card indicator comes on, CARDN does an automatic feed operation. Therefore, the last card cannot be stacker selected and cannot be both read and punched. However, the last card can be either read or punched.

Calling Sequence

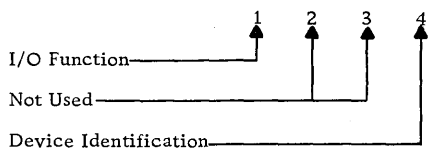
```
        LIBF    CARDN
        DC      /XXXX     (Control)
        DC      AREA      (I/O area)
        DC      00000     (Special Condition;
        .       .          not used)
        .       .
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits shown below:



I/O Function

The I/O function digit specifies a particular operation performed on the 1442 Card Read-Punch. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read	1	Control, I/O area, Special Condition**
Punch	2	Control, I/O area, Special Condition**
Feed	3	Control, Special Condition**
Stacker Select	4	Control

*Any parameter not required for a particular function must be omitted.

**Special Condition parameter may be defined as any constant.

Test. Branches to LIBF+2 if the previous operation has not been completed, or to LIBF+3 if the previous operation has been completed.

Read. Reads one card into the input area. The subroutine clears the I/O area, stores a -1 in each word in the I/O area, initiates the card operation, and returns control to the user's program. The card data is read into the user's input area on a cycle-steal basis and will be in card image or packed format depending on whether the word count is equal or unequal to 80 (see I/O Area Parameter).

NOTE: Only IBM system programs are allowed to read a //ƀ card. Any other program causing such a card to be read will cause CARDN to transfer control to EAC.

Punch. Punches into one card the number of columns of data specified by the word count found at the beginning of the user's output area. The punch operation is similar to the read operation. The character punched is the image of the leftmost 12 bits in the word.

Feed. Initiates a card feed cycle. This advances all cards in the machine to the next station, i.e., a card at the punch station advances to the stacker, a card at the read station advances to the punch station, and a card in the hopper advances to the read station. No data is read or punched as a result of a feed operation.

Stacker Select. Selects stacker 2 for the card currently at the punch station. After the card passes the punch station, it is directed to stacker 2.

Device Identification

This digit specifies which of the two possible 1442's is to be used.

0 - First 1442
1 - Second 1442

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. No chaining is permitted with the card subroutine; therefore, the control word consists of a word count only. The word count has different uses depending on whether the I/O function is Read or Punch.

Read

When a Read function is specified, the I/O area word count is used to denote the format of the input data.

Word Count	Format
=80	Card-image; data is placed in leftmost twelve bits of each of 80 words.
≠80	Packed; data in odd-numbered columns is placed in bits 8-15, even-numbered columns in bits 0-7 of each of 40 words (colums 1 and 2 form first word, 3 and 4 form second word, etc.).

Punch

When a punch function is specified the I/O area word count specifies the number of columns of data to be punched, starting with column 1.

Special Condition Parameter

(See Basic Calling Sequence.)

DISK SUBROUTINE

The disk subroutine performs all reading and writing of data relative to the 2310 Disk Storage. This includes the major functions: seek, read, and write, in conjunction with read-back check and file-protection.

The DISKN Subroutine is part of skeleton I/O. DISKN reads and writes consecutive sectors most of the time without extra disk revolutions. It is in core at all times and may be called from any interrupt level and under any mask condition. When it is called from an interrupt level which is the same or higher than the level that the disk interrupt is on, or if the disk interrupt level is masked, the DISKN subroutine operates in a masked condition utilizing indicators. Otherwise, the DISKN subroutine will operate using interrupts.

NOTE: DISKN does not save the A-register when called.

Sector Numbering and File Protection

In the interest of providing disk features permitting versatile and orderly control of disk operations, two important conventions have been adopted. They are concerned with sector-numbering and file-protection.

Successful use of the disk subroutines can be expected only if user programs are built within the framework of these conventions.

The primary concern behind the conventions is the safety of data recorded on the disk. To this end, the file-protection scheme plays a major role, but does so in a manner that is dependent upon the sector-numbering technique. The latter contributes to data safety by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs write operation. This verification requires that sector identifications be pre-recorded on each sector and that subsequent writing to the disk be done in a manner that preserves the existing identification. The disk subroutines have been organized to comply with these requirements.

Sector Numbering

The details of the numbering scheme are as follows: each disk sector is assigned an address from the sequence 0,1,..., 1599 corresponding to the sector's position in the ascending sequence of cylinder and sector numbers from cylinder 0 (outermost) sector 0, through cylinder 199 (innermost) sector 7. (The disk subroutine can address 200 cylinders, each cylinder containing eight sectors, each sector containing 321 words.)

This sector address is recorded by the user in the sector's first word, and occupies the rightmost eleven bit positions. Of these eleven positions, the three-low-order positions identify the sector (0-7) within the cylinder. Utilization of this first word for identification purposes diminishes the per sector availability of data words to 320; therefore, transmission of full sectors of data is performed in units of this amount.

File Protection

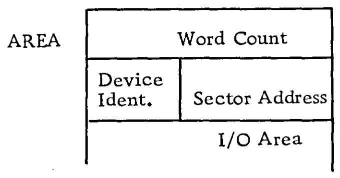
File protection is provided to guard against the inadvertent destruction of previously recorded data. By having the normal writing functions uniformly test the file-protection status of cylinders they are about to ~~to~~ write, this control can be achieved.

File protection is implemented in the TSX System by defining any disk cylinder as being either file protected or not file protected. The DUP operations are used to designate file protection.

If a cylinder is file protected, the sector addresses on that cylinder contain a one bit in bit position zero of the sector address word.

Calling Sequence

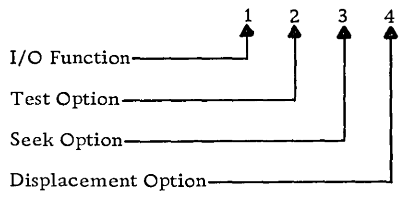
```
        LIBF      DISKN
        DC        /XXXX        (Control)
        DC        AREA         (I/O area)
        DC        00000        (Special condition;
        .         .             not used)
        .         .
        .         .
        .         .
        .         .
        .         .
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function

The I/O function digit specifies the operation to be performed on a 2310 Disk Storage. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control, I/O area
Read	1	Control, I/O area, Special Conditions**
Write without RBC	2	Control, I/O area, Special Conditions**
Write with RBC	3	Control, I/O area, Special Conditions**
Write Immediate	4	Control, I/O area
Seek	5	Control, I/O area, Special Conditions**
Seek Home	7	Control, I/O area, Special Conditions**

*Any parameter not required for a particular function must be omitted.

**Special Condition Parameter may be defined as any constant.

Test. Executes the test as specified by the test option digit. Branches to LIBF+3 if the test indicates busy, or to LIBF+4 if the test indicates not busy.

NOTE: This function requires two parameters. The I/O area parameter must be at a location other than the read or write being tested for the device busy test and must be the same location as the read or write being tested for a buffer busy test.

Read. Positions the access arm and reads data into the user's I/O area until the specified number of words have been transmitted. Although sector identification words are read and checked for agreement with expected values, they are neither transmitted to the I/O data area nor are they counted in the tally of words conveyed.

If during the reading of a sector a read check occurs, up to ten retries are attempted. If the error persists the function is discontinued, and an exit is made to EAC.

Write with Readback Check. This function first checks if the specified sector address is in a file-protected area. If it is, the subroutine exits to EAC.

If the specified sector address is not file protected, the subroutine writes the contents of the indicated I/O data area into consecutive disk sectors. Writing begins at the designated sector and continues until the specified number of words has been transmitted. A readback check is performed on the data written.

If any errors are detected, the operation is retried up to ten times. If the function cannot be accomplished by this time, an exit is made to the EAC routine.

NOTE: During a multiple sector write operation,
~~NOTE: As~~ each sector is written, the subroutine supplies the sector identification word. This word is neither obtained from the I/O area nor is it counted in the tally of words.

Write Without Readback Check. The function is the same as the function just described except that no readback check is performed.

Write Immediate. Writes data with no attempt to check for errors. This function is provided to fulfill the need for more rapid writing to the disk than is provided in the previously described Write functions. Its primary application is in the "streaming" of analog input data to the disk for temporary bulk storage.

Seek. Initiates a seek as specified by the seek option digit. If any errors are detected, the operation is retried up to 10 times.

Seek Home. This function seeks the arm to cylinder zero, ignoring sector addresses. Since all interrupt levels are masked during the operation of this function, it is recommended that it be used only for recovery from disk errors.

Test Option

If zero, the test determines if the particular device specified by the device code in the sector address is busy or not. When this test is used, do not reference a sector address parameter that might then be in use reading or writing to disk, since the sector address parameter is temporarily altered by ~~DISKIN~~ DISKN during the execution of a read or write.

If the test option parameter is one, the test determines if the I/O area referenced by the area parameter is in use by any of the disk devices.

Seek Option

If zero, a seek is executed to the cylinder whose sector address is in the disk I/O area control word: if non-zero, a seek is executed to the next cylinder toward the center, regardless of the sector address in the disk I/O area control word. This option is valid only when the seek function is specified.

NOTE: The seek function requires that the user set up the normal I/O area parameter (see I/O Area Parameter) even though only the sector address in the I/O area is used. The I/O area control word (first word) is ignored.

Displacement Option

If zero, the sector address word contains the absolute sector identification; if one, either the process or nonprocess work storage address for the specified disk (depending upon which program called DISKN) is added to the sector address word to generate the effective sector identification.

I/O Area Parameter

The I/O area parameter is the label of the first of two control words which precede the user's I/O area. The first word contains a count of the number of data words that are to be transmitted during the disk operation. This count need not be limited by sector or cylinder size, since these subroutines cross sector and cylinder boundaries, if necessary, in order to process the specified number of words.

The second word contains the sector address parameter where reading or writing is to begin. Bits 0-3 specify the device identification and bits 4-15 specify the sector address. The device identification is 0 for the first disk, 1 for the second disk, and 2 for the third disk.

Following the two control words is the user's data area. No chaining of disk I/O areas is permiteed.

Special Condition Parameter

(See Basic Calling Sequence.)

PRINTER SUBROUTINE

The printer subroutine PRNTN handles all print and carriage control functions relative to the IBM 1443 Printer. Only one line of data can be printed, or one carriage operation executed, with each call to the printer subroutine. The data in the output area must be in printer BCD form, packed two characters per computer word (See Data Codes).

The PRNTN subroutine is in core at all times as part of the skeleton if the user's system has a 1443 Printer attached. PRNTN may be called from any interrupt level or under any mask condition. When it is called for an interrupt level which is the same or higher than the level that the printer is on, or if the printer interrupt level is masked, the PRNTN subroutine operates in a masked condition utilizing indicators. Otherwise, PRNTN will operate using interrupts.

NOTE: PRNTN cannot determine if the 1443 ready indicator is on. Therefore, it is the responsibility of the operator to ensure that the ready indicator is always on. When an interrupt response is not received during on-line operation, control is transferred to EAC and a message is printed on the EAC printer.

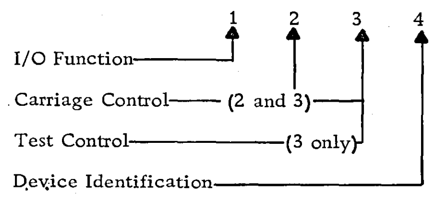
Calling Sequence

```
          LIBF        PRNTN
          DC          /XXXX        (Control)
          DC          AREA         (I/O area)
          DC          00000        (Special Con-
           .            .           ditions; not used)
           .            .
     AREA          ┌──────────────┐
                   │  Word Count  │
                   ├──────────────┤
                   │   I/O Area   │
                   │              │
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below.



I/O Function

The I/O function digit specifies a particular operation to be performed on a 1443 Printer. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Print/no checks	1	Control, I/O area
Print/with checks	2	Control, I/O area, Special Condition**
Control Carriage	3	Control
Read Control Tape	4	Control

*Any parameter not required for a particular function must be omitted.

**Special Condition Parameters may be defined as any constant.

Test. Branches to LIBF +2 if the previous operation has not been completed or to LIBF +3 if the previous operation has been completed. (Hexadecimal digit 3 determines the type of test.)

Print/No Checks. Prints characters from the user's I/O area, ignoring print checks.

Print/With Checks. Prints characters from the user's I/O area, checking for print errors. If print errors are detected, the subroutine branches to EAC. This branch occurs after the line of data has been printed.

Control Carriage. Control the carriage as specified by the carriage control digits listed in table 4.

● Table 4. Control Operations

Digit #2: Immediate Carriage Operations	
Print Functions	
0	Ignore channel 12 indicator
1	Automatically skip to channel 1 when channel 12 is detected
Control Function	
1	Immediate Skip To Channel 1
2	Immediate Skip To Channel 2
3	Immediate Skip To Channel 3
4	Immediate Skip To Channel 4
5	Immediate Skip To Channel 5
6	Immediate Skip To Channel 6
7	Immediate Skip To Channel 7
8	Immediate Skip To Channel 8
9	Immediate Skip To Channel 9
A	Immediate Skip To Channel 10
B	Immediate Skip To Channel 11
C	Immediate Skip To Channel 12
D	Immediate Space Of 1
E	Immediate Space Of 2
F	Immediate Space Of 3

Digit #3: Test and After-Print Carriage Operations	
Test Function	
0	Printer Busy
1	Message Transfer to Buffer Busy
Print Functions	
0	Space One Line After Printing
1	Suppress Space After Printing
Control Function	
1	Skip After Print To Channel 1
2	Skip After Print To Channel 2
3	Skip After Print To Channel 3
4	Skip After Print To Channel 4
5	Skip After Print To Channel 5
6	Skip After Print To Channel 6
7	Skip After Print To Channel 7
8	Skip After Print To Channel 8
9	Skip After Print To Channel 9
A	Skip After Print To Channel 10
B	Skip After Print To Channel 11
C	Skip After Print To Channel 12
D	Space 1 After Print
E	Space 2 After Print
F	Space 3 After Print

Read Carriage Control Tape. The A-register is set as follows for the respective conditions. If the 1443 is busy at the time of the call, PRNTN will wait until it is not busy before reading the carriage control tape.

A-register	Condition
Bit 0 set to 1	If control tape channel 9 is on.
Bit 1 set to 1	If control tape channel 12 is on.
Bit 2 set to 1	If control tape channel 1 is on.

Carriage Control

Digits 2 and 3 specify the carriage control functions listed in Table 4. An immediate request is executed before the next print operation; an after print request is executed after the next print operation and replaces the normal space operation.

If the I/O function is print, digits 2 and 3 are examined; if the I/O function is control, and digits 2 and 3 both specify carriage operations, only digit 2 is used.

Test Control

Digit 3 determines the type of test. It is shown here and repeated in Table 4.

Test Function

0 - Printer busy

1 - Message transfer to buffer busy

Device Identification

This digit must be 0.

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. No chaining is permitted with the printer subroutines; therefore the control word consists of a word count only. The word count specifies the number of computer words of data to be printed. The data must be in printer BCD format, packed two characters per computer word.

Special Condition Parameter

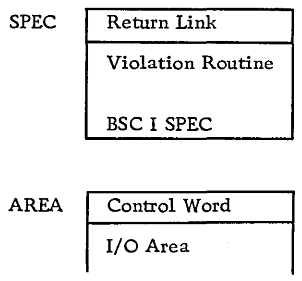
(See Basic Calling Sequence.)

MAGNETIC TAPE SUBROUTINE

The magnetic tape subroutine performs all read, write, and control functions relative to the IBM 2401 and 2402 Magnetic Tape Units.

Calling Sequence

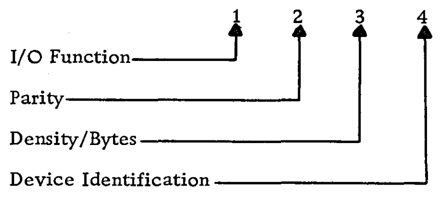
LIBF	MAGT	
DC	/xxxx	(Control)
DC	AREA	(I/O area)
DC	SPEC	(Special condition routine)



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function

The I/O function digit specifies the operation to be performed on a magnetic tape unit. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Hexadecimal Value	Required Parameters*
Test	0	Control
Read/with error retries	1	Control, I/O Area Special Condition**
Read/without error retries	2	Control, I/O Area, Special Condition**
Write/with error retries	3	Control, I/O Area Special Condition**
Write/without error retries	4	Control, I/O Area Special Condition**
Rewind	5	Control
Rewind and unload	6	Control
Backspace	7	Control
Write tape mark	8	Control
Erase	9	Control
Reset	A	Control

*Any parameter not required for a particular function must be omitted. All required parameters for a particular function must be present.

**Special Condition parameter is the label of the user's routine to process intermediate interrupts, error conditions, etc.

Test. Branches to LIBF +2 if the previous operation has not been completed or to LIBF +3 if the previous has been completed.

Read/with error retries. Reads the next record into the I/O area. If a read check occurs, the subroutine retries the operation up to 100 times. Each attempt includes backspacing the tape one record and then re-reading the record. Every third attempt, the tape is backspaced three records and then spaced forward twice before re-reading begins. If at any time, before 100 retries have been attempted, the record is read correctly, the subroutine exits from the retries as if no error had occurred.

If a read check still exists after 100 attempts to re-read, the subroutine exits to EAC, then to the user via the special condition parameter, then clears the routine Busy indicator, and exits normally.

If intermediate end-of-table interrupts are sensed, the subroutine exits to the user via the special condition parameter, with a code of 96 in the A-register. This occurs for every such interrupt on each attempt to re-read the record.

Read/without error retries. Reads the next record into the I/O area. No error correction is attempted on error conditions. This read mode is useful when the end-of-table interrupts are used to process data already transferred to core storage from tape during the read operation. It is the user's responsibility to perform any error correction desired.

Write/with error retries. Writes the requested number of words from the I/O area as one record on the specified tape. If a write check is detected, the tape is backspaced, several inches of tape are erased, and the write operation is restarted. If a write check still occurs after three attempts, the subroutine exits to EAC, then to the user via the special condition parameter, then exits normally.

If intermediate end-of-table interrupts are sensed, the subroutine exits to the user via the special condition parameter, with a code of 96 in the A-register. This occurs for every such interrupt on each attempt to write the record.

Write/without error retries. Writes the requested number of words from the I/O area as one record on the specified tape. No error correction is attempted if a write check occurs. This write mode is useful when streaming high speed inputs onto tape for bulk storage.

Rewind. Initiates a tape rewind and returns control to the user.

Rewind and unload. Initiates a tape rewind and unload and returns control to the user.

Backspace. Backspaces one record. If the tape is at load point, no backspace occurs.

Write tape mark. Writes a tape mark on the tape.

Erase. Erases approximately three and one-half inches of tape.

Reset. Stops the magnetic tape operation in progress and resets all indicators.

Parity

The parity digit specifies the parity of the records that are being read or written. A zero specifies odd-bit parity and a one specifies even-bit parity. This digit is used only for read and write functions on 7-track tape and is ignored for 9-track magnetic tape.

When writing 7-track tape in even parity, an all zero (000000) or A bit (010000) character will be written as an A bit (010000). When reading 7-track tape in odd parity, an A (010000) tape character will be read as an all-zero (000000) character. What bits in core are read as a character on tape depends on the byte mode (see IBM 1800 Data Acquisition and Control System Data Processing Input/Output Units, Form A26-5969.

Density/Bytes

This digit specifies the density and the number of tape bytes (6-bit characters) per word to be used with 7-track magnetic tape. It is ignored for 9-track operations.

Bits zero and one specify the number of bytes per word:

00	Three bytes
01	Two bytes

Bits two and three specify the density:

00	800 bits/inch
01	200 bits/inch
10	556 bits/inch

Device Identification

This digit specifies which of the two possible magnetic tape units is to be used. The units may not operate simultaneously.

0	First unit
1	Second unit

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's first I/O area. This control word consists of a word count, chaining indicator, and end-of-table interrupt indicator. The word count must not be less than 8 characters for a read and 9 characters for a write. For a complete description of this word, see IBM 1800 Data Acquisition and Control System Functional Characteristics Manual (Form A26-5918).

End-of-table interrupts are recognized by the subroutine and control is given to the user via the special condition parameter, with a code of 0600 in the A-register. Note that on error retries, every end-of-table interrupt on each attempt to read or write, the record is recognized.

Special Condition Parameter

The special condition parameter is the label of the user's routine to which control is transferred according to Table 5. The user's routine is entered with a BSI instruction and should return control to the magnetic tape subroutine at the next location following the call.

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must also be in the skeleton.

● Table 5. Magnetic Tape Special Conditions

Contents of A-register	Contents of Q-register	Reason	Notes
0600	XXXX	End-of-table Interrupt	An end-of-table interrupt has occurred for the current record being read or written.
0001	XXXX	End-of-file on read	An end-of-file mark has been read. The subroutine will clear busy and exit normally when the user returns control.
0002	Channel word count	Wrong length record read, no read checks	The wrong length record indicator was on for this read. The subroutine will clear busy and exit normally when the user returns control.
0003	XXXX	Intermediate read or write check	Read or write checks have occurred on the last record. This exit occurs only if the read/ or write/ with retries was specified. The subroutine will continue when the user returns control.
0004	Channel word count	Read or write checks on last record	Read or write checks have occurred on the last record. This exit occurs when the error correction procedure has failed to clear up the problem if read/ or write/ without retries has been specified. The subroutine will clear busy and exit normally when the user returns control. The subroutine does not check for noise records and the user must determine the length of the record read or written.

End-of-Tape

When an end-of-tape condition is detected, the subroutine takes the following action:

For read functions, the subroutine executes a rewind and unload instruction, then exits normally.

For a write with error retries, the subroutine writes a tape mark, then executes a rewind and unload instruction, then exits normally.

Sensing of an end-of-tape condition terminates any error recovery procedure.

Error Count

The subroutine maintains an error count of the number of times EAC has been called. When the count exceeds 50, the subroutine calls EAC for an excessive tape error message. The error count is set to zero whenever a tape is rewound.

PRINTER-KEYBOARD SUBROUTINE

There is one subroutine for the transfer of data to and from the IBM 1053 and the 1816 Printer-Keyboard. It can be called by either of its two names: TYPEN or WRTYN.

If the system configuration does not include the 1816 Printer-Keyboard, the keyboard portion of the subroutine is removed (origined out).

Message Storage

At TASK assembly time, the user must define two things for the TYPEN subroutine: Whether messages to the 1053 are to be buffered to disk or not, and the message unit size required for each 1053.

If buffering of messages to disk has been defined, the size of any message can be greater than the size of the message unit. If no buffering has been defined, the message can never be of a greater size than that defined for the message unit.

Buffering of Messages to Disk

The maximum core size that may be specified for a message buffer is 319 words, and the minimum core size is 20 words. Each buffer can be a different size. The optimum size is between 40 and 80 words (80 to 160 characters).

When a 1053 or 1816 printer is busy and another call to the printer is given, the excess message data is temporarily stored in a disk buffer. (The data of other additional calls will be placed behind the first call.) When the current message (or portion of a message) has been completed, the longest stored (timewise) disk message is brought into core and printing continues (see Message Priority). If a call is given to a printer that is not busy and the entire message fits into the core buffer, no buffering to disk occurs.

If TYPEN/WRTYN is called from an interrupt level higher than or the same as the interrupt level for the 1053, or the 1053 interrupt level is masked, the message will be stored on disk, but no output will take place on the 1053 until operation has been restored to an interrupt level that is lower than that of the 1053. If the disk buffer is full in the case just described, the message being printed will be temporarily discontinued, while the new message is typed by using indicators rather than using the interrupts.

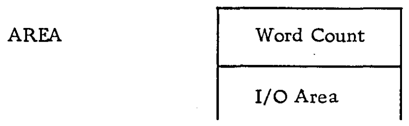
Non-Buffering of Messages to Disk

There is no maximum to the message unit size specified. The minimum size is 20 words, except if the user plans to use FORTRAN I/O, in which case the minimum size is 80 words.

When non-buffering of messages to disk has been defined, the user should not call to print a message from an interrupt level that is higher than or equal to the priority of the typewriter interrupt level, or call to type a message if the interrupt level of the typewriter is masked. If such calls are executed, the present message will be temporarily discontinued, while the new message is typed by using indicators instead of the interrupts. TYPEN moves the user's message into the buffer in the typewriter device table in core before returning to the user, therefore, the user's message area is free as soon as a return from TYPEN has been executed.

Calling Sequence

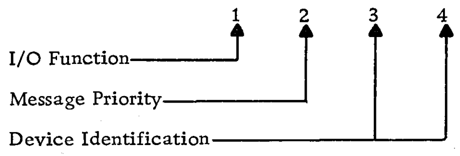
LIBF	TYPEN or WRTYN	
DC	/xxxx	(Control)
DC	AREA	(I/O area)
DC	00000	(Special Condition; not used)



The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function

The I/O function digit specifies a particular operation to be performed on an 1816 Printer-Keyboard and/or 1053 Printers. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read-Print	1	Control, I/O area, Special Condition**
Print	2	Control, I/O area, Special Condition**

*Any parameter not required for a particular function must be omitted.

**Special Condition parameter may be defined as any constant.

Test: Branches to LIBF +2 if the previous operation has not been completed or to LIBF +3 if the previous operation has been completed.

Read-Print: Reads from a keyboard and prints on specified 1053 printer the requested number of characters. The operation sequence is as follows:

1. The calling sequence is analyzed by the call routine which then unlocks the keyboard.
2. When a key is pressed, a character interrupt signals the interrupt response routine that a character is ready to be read into core storage.
3. The interrupt response routine converts the keyboard data to the typewriter output code (see Data Codes), printing each character on the specified printers as the character is read and unlocking the keyboard for entry of the next character, if any.
4. The printer interrupt occurs whenever the specified printer has completed its print operation.
5. Items 2 to 4 are repeated until the specified number of characters has been read and printed or an EOF character is entered. The characters read into the I/O area are in IBM card code; that is, each 12-bit image is left-justified in one 16-bit word.

Three keyboard functions are recognized by the printer-keyboard subroutine:

Backspace (ER CHR). The operator presses the backspace key whenever the previous character is in error. The interrupt response routine, sensing the control character, backspaces the specified printer and prints a slash (/) through the previous character. In addition, the routine prepares to replace the previous character in the I/O area with the next character.

Re-entry (ER FLD). When the interrupt response routine recognizes the re-entry control character, it assumes that the entire message is in error and is to be re-entered. The routine restores the carrier to a new line. In addition, the routine prepares to replace the old message in the I/O area with the new message.

End-of-Message (EOF). When the interrupt response routine recognizes the end-of-message control character, it assumes the message has been completed, stores an NL character in the I/O area, and terminates the operations.

NOTE: There will be no NL character in the I/O area if the message was terminated because the word count went to zero.

Print: Prints the specified number of characters on the specified printer. Printer interrupts occur when the specified printer has completed a print operation. When the interrupt has been received, the character count is checked. If the specified number of characters has not been printed, printing is initiated for the next character. This sequence continues until the specified number of characters has been printed. Data to be printed must be in typewriter-output code (see Data Codes) packed two characters per 16-bit word. Control characters can be embedded in the message wherever desired.
NOTE: Printing starts wherever the printing element is positioned; that is, a carrier return to a new line is not automatic when the subroutine is called. Do not press carrier RETURN with the motor off (it times out in two minutes) as it causes the TSX system to loop when the 1816 is selected.
The two words preceding the beginning of the I/O area may be altered temporarily if the message is buffered to disk. Therefore, these words should never be storage protected, nor should they be executable instructions in an interrupt routine.

Message Priority

The message priority digit is used to denote the type of message to be printed: (0) normal or (1) priority. Messages are stacked (if necessary) on the disk and are printed when those that precede it have been printed. Priority messages are also stacked if the printer is busy at the time of the call, but as soon as the message unit in the core buffer area is completed, printing of the priority message begins. Message priority has no meaning on those systems that do not buffer messages to disk.

Note that although a message can be of any length, when a priority call is made, a normal message can be interrupted at the place(s) it overflows the core buffer and becomes stored on disk.

Printing of any stacked normal messages resumes at the completion of the priority message printing.

NOTE: When there is buffering of messages to disk, there are, in effect, four priorities by which messages are printed:

1. EAC messages
2. Keyboard entries
3. Priority messages
4. Normal messages

When there is no buffering of messages, there are, in effect, two priorities of messages:

1. EAC messages
2. Keyboard, priority, and normal messages.

Device Identification

Two groups can be controlled by the printer-keyboard subroutine: the basic group (which includes the primary 1816 or 1053) and the output printer expander group. Each group may include up to four 1053 Printers, or one 1816 Printer-Keyboard and up to three 1053 Printers.

Hexadecimal digit #4 identifies which 1816/1053 is to print the message on a print function and which 1816/1053 is to print the keyboard entry for visual verification on a read function. Hexadecimal digit #3 is used only on a test function. If digit #3 is a 1, it indicates that a busy test is to be made for the keyboard on the same group as the 1816/1053 specified in hexadecimal digit #4. The possible combinations for the test and read/write functions are listed below.

Test		Read/Write*	Device
Hexadecimal Digit #3	Hexadecimal Digit #4	Hexadecimal Digit #4	
0	1	1	1816/1053-1, Group 1
0	2	2	1053-2, Group 1
0	3	3	1053-3, Group 1
0	4	4	1053-4, Group 1
1	1, 2, 3, 4		Keyboard on Group 1
0	5	5	1816/1053-1, Group 2
0	6	6	1053-2, Group 2
0	7	7	1053-3, Group 3
0	8	8	1053-4, Group 4
1	5, 6, 7, 8		Keyboard on Group 2

*Hexadecimal digit #3 is ignored on a read/write instruction. The keyboard being used for input (read) will be on the same group as the printer selected to type the input message for visual verification, i.e., if hexadecimal digit #4 for a read/write instruction is a number between 1 and 4, the keyboard on printer group 1 must be used for input.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. Chaining is not permitted; therefore, the control word consists of a word count only. The word count specifies the number of words to be read into or printed from.

This word count is equal to the number of characters if the read-print function is requested but is equal to one-half the number of characters if the print function is requested.

Special Condition Parameter

(See Basic Calling Sequence.)

Operator Request Function

By pressing the operator request key on the keyboard, the operator can inform the program that he wants to enter some data from the keyboard. The interrupt that results from such a request causes the Keyboard-Printer subroutine to program an interrupt to a lower level. The user must specify the level at task assembly time, and provide an interrupt program to service the programmed interrupt. This precludes the user from having any other programmed level interrupts on this level. Word 149 of core is set to a negative value if the request is from the 1816 in the first group of 1053/1816s, and is set to a positive value if the request came from the 1816 in the second group.
An 1816 keyboard entry should be dependent on an interrupt from the keyboard request key. The 1816 is not considered a process I/O device

PAPER TAPE SUBROUTINE

The paper tape subroutine handles the transfer of data from the IBM 1054 Paper Tape Reader to core storage and from core storage to the IBM 1055 Paper Tape Punch. Any number of characters can be transferred with one calling sequence.

When called, the paper tape subroutine starts the reader or punch and then, as interrupts occur, transfers data to or from the user's I/O area. The data is packed two characters per computer word by the subroutine when reading, and must be in that form when the subroutine is called for a punch function.

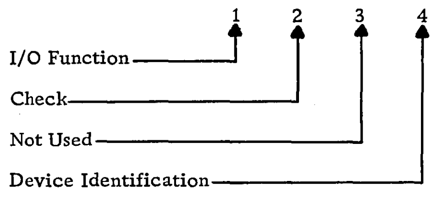
Calling Sequence

```
        LIBF        PAPTN
        DC          /xxxx       (Control)
        DC          AREA        (I/O area)
        DC          00000       (Special Condition; not used)
         .            .
         .            .
         .            .
         .            .
AREA          ┌──────────────┐
              │  Word Count  │
              ├──────────────┤
              │  I/O Area    │
              ¦              ¦
              ¦              ¦
```

The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function

The I/O function digit specifies a particular operation to be performed on a 1054/1055 Paper Tape attachment. The functions, their associated digital value, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read	1	Control, I/O area, Special Condition**
Punch	2	Control, I/O area, Special condition**

*Any parameter not required for a particular function must be omitted.

**Special Condition parameter may be defined as any constant.

Test. Branches to LIBF+2 if the previous operation has not been completed or to LIBF+3 if the previous operation has been completed.

Read. Reads paper tape characters into the specified number of words in the I/O area. Initiating read motion causes an interrupt to occur when a character can be read into core. If the specified number of words has not been filled, or the stop character has not been read (see Check), reader motion is again initiated.

Punch. Punches paper tape characters into the tape from the words in the I/O area. Each character punched causes an interrupt which indicates that the next character can be accepted. The operation is terminated either by encountering a stop character or by the total number of words having been transferred. (See Check.)

Check

The check digit specifies whether or not checking is desired while doing a read or punch operation.

0 — Check

1 — No check

No Check. The read or punch function is terminated when the specified number of words has been read or punched, two characters per word. No check is made for a delete or stop character.

Check. This function should be used with the perforated tape and transmission (PTTC/8) code only (see Data Codes). The PTTC/8 code for DEL will be used as the delete character when doing a Read. The delete character is not placed in the I/O area and therefore does not enter into the count of the total number of words to be filled.

The PTTC/8 code for NL will be used as the Stop character when doing a read or punch. On a read operation, the stop character is transferred into the I/O area and causes the operation to be terminated. On a punch operation, the stop character is punched in the paper tape and causes the operation to be terminated.

When the stop character is encountered before the specified number of words has been read or punched, the operation is terminated. When the specified number of words has been read or punched, the operation is terminated even though a stop character has not been encountered.

Device Identification

Since only one 1054/1055 Paper Tape attachment is supported by the paper tape subroutines, the device identification code is not needed to indicate which 1054/1055 device the user is specifying, nor is the identification code needed to identify whether the reader or the punch is being called, since the I/O function is sufficient for this determination. However, when the Test function is specified, the PAPTN subroutine must be told which device (reader or punch) is to be tested for an "operation complete" indication. (Remember that both the reader and the punch can operate simultaneously.) If it is a 0, the subroutine tests for a "reader complete" indication; if it is a 1, the subroutine tests for a "punch complete" indication.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. Chaining is not permitted with the paper tape subroutine; therefore, the control word consists of a word count only. The word count specifies the number of words to be read into or punched from. Since characters are packed two per word in the I/O area, this count is one-half the maximum number of characters transferred. Because an entire 8-bit channel image is transferred by the subroutine, any combination of channel punches is acceptable. The data may be a binary value or a character code. The code most often used is the PTTC/8 code (see Data Codes).

Special Condition Parameter

(See Basic Calling Sequence.)

PLOTTER SUBROUTINE (PLOTX)

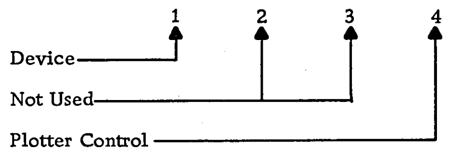
The PLOT X subroutine converts the hexadecimal digit in the parameter into a control word. The control word is stored in a buffer inside the PLOTX subroutine. One digit is transferred with each calling sequence. When the plotter is ready to accept controls, the movement of the plotter recording pen is controlled by the words in the PLOTX buffer.

Calling Sequence

```
        LIBF        PLOTX
        DC          /XXXX      (Control)
```

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below.



Device

The device identification digit must be a number between 0-7.

Control Plotter

The control plotter digit specifies the recording pen action to be taken. This digit is expressed in hexadecimal.

Hexadecimal Digit	Plotter Action
0	Pen down
1	Line segment = +Y
2	Line segment = +X, +Y
3	Line segment = +X
4	Line segment = +X, -Y
5	Line segment = -Y
6	Line segment = -X, -Y
7	Line segment = -X
8	Line segment = -X, +Y
9	Pen up
A-F	Not used

If there is no room to put the control digit into the buffer, the routine will loop until there is room.

DIGITAL INPUT SUBROUTINE

This subroutine is used to read in or to check digital information for a number of digital input groups under either direct program control or data channel control. Table chaining is permitted when the data channel is specified, however, continuous scanning is not permitted.

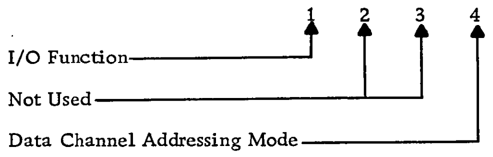
Calling Sequence

```
        LIBF    DINP
        DC      /xxxx     (Control)
        DC      AREA      (I/O Area-DI address)
        DC      SPEC      (Special condition routine)
        .       .
        .       .
        .       .
        .       .
SPEC    | Return Link       |
        | Violation Routine |
        | BSC I SPEC        |

AREA    | Control Word      |
        | DI Address        |
        | I/O Area          |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, used as shown below:



I/O Function

The I/O function digit specifies the operation to be performed by the digital input subroutine. The functions, associated digital value, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read Data Channel	1	Control, I/O Area, Special Condition
Read Sequential Program Control	2	Control, I/O Area, Special Condition
Read One Point Into A-Reg.	3	Control, I/O Area, Special Condition
Reset	4	Control

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed, or to LIBF+3 if the previous operation has been completed.

Read Data Channel. Initiates the operation and returns control to the user. A table-complete interrupt occurs after the last word of a data table has been processed. (The user can suppress this interrupt by placing a 1 in bit position 1 of the I/O area control word. However, the subroutine requires the interrupt on the last table in the chain and will force the interrupt by clearing bit position 1 of the control word for the last table.) Whenever this interrupt occurs (except on the last table), the subroutine branches indirect via the address specified in the special condition parameter, with a code of 73 in the A-register. The subroutine is "busy" until all of the data has been read, and the last table interrupt has occurred.

NOTE: Continuous scan of digital input is not allowed, since this can lock out the CPU.

Read Sequential-Program Control. Reads in the requested number of digital input values. After each value is read, the word count is checked. If the requested number of values has not been read, the subroutine reads in the next value. When the requested number has been read, the subroutine returns control to the user. There are no interrupts involved in the operation, and the subroutine does not return control to the user until the entire operation is complete. The special condition parameter is not used; it can be any data.

Read One Point Into A-Register. Reads the point specified in the area parameter and places the value in the A-register. The special condition parameter is not used; it can be any data.

Reset. Halts all digital input operations in progress and resets all indicators.

Data Channel Addressing Mode

The addressing mode digit specified one of six addressing options available. The addressing mode digit is used only if the function is read data channel.

0 — Random addressing

1 — Sequential addressing

2 — Single addressing

3 — Random addressing with external synchronization

4 — Sequential addressing with external synchronization

5 — Single addressing with external synchronization

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. Since chaining is permitted (data channel control only), the chaining bit and the suppress interrupt indicator bit are examined by the subroutine.

If single or sequential addressing mode is specified, the word count is the number of groups (words) to be read, plus one, since the count includes the digital input address word, which contains the address of the initial digital input group to be read. If single addressing mode is specified, the initial input group is read repeatedly into successive I/O area locations. If the sequential addressing mode is specified, the initial group address is incremented after each data word is read.

If the random addressing mode is specified, the word count is twice the number of groups (words) to be read. Each digital address precedes its data word.

For the function of reading one point into the A-register, the I/O parameter is the address of the digital input group.

Special Condition Parameter

(See Functions Used.)

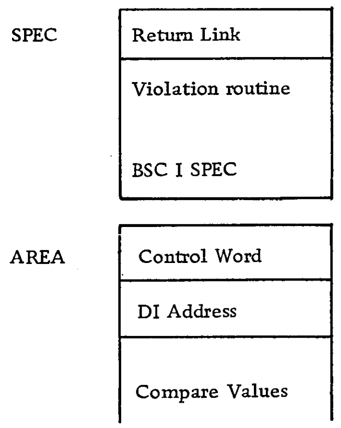
If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must be in the skeleton.

DIGITAL INPUT READ/COMPARE

This subroutine reads in a number of values under program control and compares them with a table of values supplied by the user. If any value read is not equal to the user-supplied value, the subroutine branches to a user's violation subroutine. There is no interrupt associated with the routine; therefore, it may be in core with either of the other digital input routines.

Calling Sequence

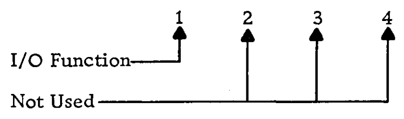
```
        LIBF    DICMP
        DC      /xxxx     (Control)
        DC      AREA      (I/O area)
        DC       SPEC     (Special Conditions
                           Routine)
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the function to be performed. The functions, associated digital values, and the required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read/ Compare	1	Control, I/O Area, Special Condition

*Any parameter not required for a function must be omitted.

Test. Branches to LIBF+2 if the previous operation has not been completed, or to LIBF+3 if the previous operation has been completed.

Read and Compare. Reads in, under direct program control, compares with the number of words in the user's I/O area. Each group is read in separately and compared with one word in the I/O area. If the two are not equal, the subroutine branches indirect via the special condition parameter.

When an error exit occurs, the A-register contains the digital input address of the group in error; the Q-register contains the current reading that did not compare with the word in the I/O area. After all the specified comparisons have been made, the subroutine exits normally.

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the address of the first group and the values compared. The control word specifies the word count of the number of values (words) to be compared, plus one, for the address of the initial digital input group to be read.

Special Condition Parameter

This parameter specifies the address of the violation routine to be executed when a comparison of unequal values results.

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must also be in the skeleton.

DIGITAL INPUT READ/EXPAND

This subroutine reads in a digital input value and expands it into 2, 4, 8, or 16 words. It provides the FORTRAN programmer with the ability to put each of the different digital input values into a number of words, with 8, 4, 2, or 1 bit per word. (All output values are right-justified.) There is no interrupt associated with the routine; therefore, it can be in core with any of the other digital input routines.

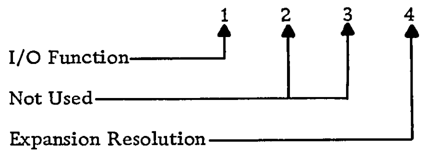
Calling Sequence

```
        LIBF    DIEXP
        DC      /xxxx     (Control)
        DC      AREA      (I/O Area)
        DC      00000     (Special Condition;
                            not used)
         .        .
         .        .
         .        .
         .        .
         .        .
AREA   | DI Address |
       | Input      |
       | Area       |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the type of operation to be performed. The functions, associated digit values, and required parameters are listed and described below.

Function	Digit Value	Required Parameter*
Test	0	Control
Read/ Expand	1	Control, I/O Area, Special Condition

*Any parameter not required for an operation must be omitted.

Test. Branches to LIBF+2 if the data channel is busy, or to LIBF+3 if it is not busy.

Read/Expand. Reads the digital input value specified by the area address and separates the value into the number of bits per word specified by the expansion resolution digit.

Expansion Resolution

This digit specifies one of four resolutions.

1 — 16 Words, 1 bit per word

2 — 8 Words, 2 bits per word

4 — 4 Words, 4 bits per word

8 — 2 Words, 8 bits per word

I/O Area Parameter

The I/O area parameter is the label of the DI address that precedes the input area. The first word of the I/O area contains the DI address. High-order bits of the DI value (bit 0, etc.) will be stored in the second word of the I/O area, while low-order bits (bit 15) will be stored in the last word of the I/O area. All stored values are right-justified.

Special Condition Parameter

(See Basic Calling Sequence.)

DIGITAL/ANALOG OUTPUT SUBROUTINE

This subroutine transfers digital/analog information to a number of addresses, under direct program control or data channel control. Table-chaining is permitted on the data channel; however, continuous scanning is not permitted.

Calling Sequence

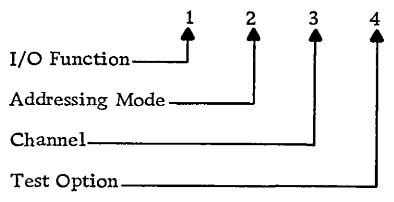
```
        LIBF    DAOP

        DC      /xxxx     (Control)

        DC      AREA      (I/O Area)

        DC      SPEC      (Special Condition
                           Routine)
```

SPEC	Return Link
	Violation Routine BSC I SPEC
AREA	Control Word
	Address
	I/O Area

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits used as shown below:



I/O Function

The I/O function digit specifies the operation to be performed by the digital/analog output subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Write	1	Control, I/O Area, Special Condition
Write Pulse	2	Control, I/O Area, Special Condition
Write Buffered	3	Control, I/O Area, Special Condition
Reset	4	Control

*Any parameter not required for a particular function must be omitted.

Test. Depending on the test option, branches to LIBF+2 if test indicates busy, or to LIBF+3 if test indicates not busy.

Write. Writes the requested number of digital/ analog values. If direct program control is specified, no interrupts are involved in the operation. Therefore, the subroutine does not return control to the user until the entire operation is complete. After each value is written, when using direct program control, the word count is checked. If the requested number of values has not been written, the subroutine writes the next value. If the requested number has been written, the subroutine returns control to the user. This routine only operates in the sequential mode. The DAO address is incremented by one for each value written.

If the data channel is specified, the subroutine initiates the operation and returns control to the user. Table-complete interrupts occur after the last word of a data table has been processed. (The user can suppress this interrupt by placing a 1 in bit position 1 of the I/O area control word. However, the subroutine requires this interrupt on the last table in the chain and will force the interrupt by clearing bit position 1 of the control word for the last table.) Whenever these interrupts occur (except on the last table), the subroutine branches to the address specified in the special condition parameter with a code of 82 in the A-register. The subroutine is "busy" until all of the data has been written and the last table interrupt has occurred.

NOTE: Continuous output is not allowed on DAOP, since this can lock out the P-C.

Write Pulse. Same as write, except that the control for pulse output is given after the write function has been performed.

Write Buffered. Same as write, except that the control for buffered output is given after the write function has been performed.

Reset. Resets all digital/analog output operations in progress and resets all indicators.

Addressing Mode

The addressing mode digit specifies one of four addressing options for data channel operations only.

0 - Random addressing

1 - Single addressing

2 - Random addressing with external synchronization

3 - Single addressing with external synchronization

Channel

This digit specifies the method of data transfer used for this operation.

0 - Direct program control

1 - Data channel control

Test Option

If zero (0), DAOP tests to see if the previous call has been completed. If one (1), it tests to see if the pulse output timer is on.

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. If the function is direct program control, the word count is the number of points to be written, plus one for the first DAO address. If the function is data channel control, the mode determines what the word count should be. If the mode specifies random addressing, the table contains interleaved digital/analog addresses and data to be written. Each address precedes its data word. If the mode specifies single addressing, the table contains one address followed by all the data words to be written for that address.

The word count is equal to the number of address words and data words in the table. If the mode indicates random addressing, the word count is twice the number of data words to be written, since there is an address for each data word. If the mode indicates single addressing, the word count is the number of data words to be written plus one, since there is only one address word. The subroutine expects the 16-bit digital value for output to be in the following format.

Bits	0	Sign
	1-13	Data bits (DAC Models 3 and 4)
	1-10	Data bits (DAC Models 1 and 2)

Special Condition Parameter

(See Functions Used)

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must be in the skeleton.

ANALOG INPUT — SEQUENTIAL SUBROUTINE

This subroutine is used to read analog data (relay or solid state) for a number of sequential points under direct program control or data channel control. The subroutine can be in core storage at the same time as the AIPTN and AIRN subroutines, but the only analog subroutine with which it can operate simultaneously is the AIPTN, and only if the multiplexer overlap feature is attached and the AIPTN subroutine is reading in a relay value. Within a table, relay data and solid state data cannot be mixed for any one operation; however, relay tables and solid state tables can be interleaved by chaining the tables together.

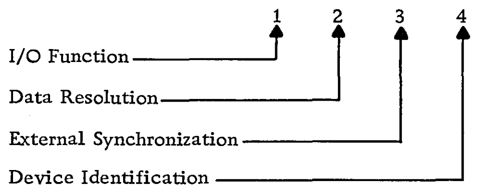
Calling Sequence

```
        LIBF    AISQN
        DC      /xxxx       (Control)
        DC      AREA        (I/O Area)
        DC      SPC         (Special Condition)
        .       .
        .       .
SPC    | Return Link          |
       | Error Routine        |
       | BSC I    SPC         |
        .       .
        .       .
        .       .
AREA   | Control Word         |
       | Multiplexer Address  |
       | I/O Area             |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits as shown below:



I/O Function

The I/O function digit specifies the operation to be performed by the AISQN subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read-Direct Program Control	1	Control, I/O Area, Special Condition
Read-Data Channel Control	2	Control, I/O Area, Special Condition
Read-Data Channel Control Without Error Retry	3	Control, I/O Area, Special Condition
Reset	4	Control
Read and Transfer	5	Control, I/O Area, Special Condition

*Any parameter not required for a particular function must be omitted.

Test. Branches to LIBF+2 if the analog feature is busy, or to LIBF+3 if the analog feature is not busy.

Read-Direct Program Control. Under direct program control, reads the specified number of analog point values into the I/O area. An interrupt occurs when a value has been converted and can be read into core storage. If the number of points specified has not been read, another sequential point read operation is initiated. If the number of points specified has been read, no more read operations are initiated and no more interrupts will occur. The special condition parameter is not used; it can be any data.

Read-Data Channel Control. Under data channel control, reads the specified number of analog point values into the I/O area tables. Table-complete interrupts occur after the last word of a data table has been processed. (The user can suppress this interrupt by placing a 1 in bit position 1 of the I/O area control word. However, the subroutine requires this interrupt on the last table of the chain and will force the interrupt by clearing bit position 1 of the control word for the last table.) Whenever these interrupts occur (except on the last table), the subroutine branches to the address specified in the special condition parameter with a code of 65 in the A-register. The subroutine is "busy" until all of the data has been read and the last table interrupt has occurred. If a continuous scan is performed, both the AISQN subroutine and the data area must be in skeleton.

Read-Data Channel Control Without Error Retry. This is the same as read-data channel control, except that on an error, no retries will be attempted. If an error occurs, AISQN will branch to the user's special condition routine with an error code of 65 in the A-register. The user is expected to return to AISQN to continue.

Reset. Halts all analog input operations in progress and resets all indicators.

Read and Transfer. Same as read-direct program control, except that after each ADC value is read into core, the subroutine branches to a user-specified special condition routine. The multiplexer address is in the A-register and the core location of the read value is in the Q-register. The user must return to AISQN after performing conversion.

Data Resolution

This digit indicates one of three data-resolution options.

0 - Standard, 11 bits, plus sign

1 - High, 14 bits, plus sign

2 - Low, 8 bits, plus sign

External Synchronization

This digit indicates whether external synchronization is desired.

0 - No

1 - Yes

Device Identification

This digit indicates one of two ADC's.

0 - First ADC

1 - Second ADC

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. If chaining is desired and the read-data channel control function has been requested, the user must set chaining bits and suppress interrupt indicator bits; otherwise, the control word contains a word count only. The word count is one greater than the number of ADC values read since it must include the multiplexer address as well as the data words.

The multiplexer address word follows the control word and contains the address of the first point to be read. The number of points to be read is one less than the word count in the control word. The address word has the following format.

Bits 0-2	Not used
3	1 for solid state; 0 for relay
4-5	Not used
6-15	Multiplexer point address

The subroutine reads in 16-bit values having the following format.

Bits 0	Sign
1-14	Data bits (1-8 if low, 1-11 if standard, 1-14 if high)
15	Overload indicator

An example of an I/O area used to read in the value of 15 sequential solid-state analog points, starting with an analog point address of 6, is shown below.

```
AREA  +0 DC    10 (word count)
      +1 DC    /1006 (multiplexer address word)
      +2 DC    0 (register for value of point #6)
      +3 DC    0 (register for value of point #7)
        .        .
        .        .
        .        .
        .        .
      +16 DC   0 (register for value of point #20)
```

Special Condition Parameter

(See Functions Used.)

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must be in the skeleton.

ANALOG INPUT – SINGLE READ SUBROUTINE

This subroutine has three major uses: (1) to read analog data if there is no data channel for analog, (2) to read a single point even if there is a data channel for analog, and (3) (if the multiplexer overlap feature is on the system) to provide that form of relay overlap in which a single relay conversion is started and then overlapped by a series of AISQN subroutine conversions. The AIPTN subroutine can be in core at the same time as the AISQN or AIRN subroutines; all three subroutines use ANINT to process interrupts. Both ADCs can be operated simultaneously.

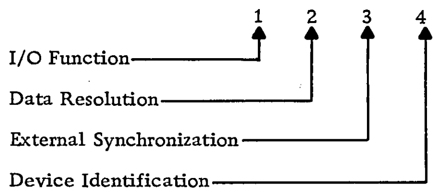
Calling Sequence

```
        LIBF    AIPTN
        DC      /xxxx     (Control)
        DC      AREA      (I/O Area)
        DC      POINT     (Point Address)
        DC      00000     (Special Condition)
        .       .
        .       .
        .       .
        .       .
        .       .
AREA  1-Word Input Area
        .       .
        .       .
        .       .
POINT  Multiplexer Address
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specified the operation to be performed by the AIPTN subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read	1	Control, I/O Area, Point Address, Special Condition
Reset	3	Control

*Any parameter not required for a function must be omitted.

Test. Branches to LIBF +2 if the analog feature is busy, or to LIBF +3 if the analog feature is not busy.

Read. Reads in the converted analog value for the specified multiplexer point and places it in the user's 1-word input area.

Reset. Halts all analog input operations in progress and resets all indicators.

Data Resolution

This character indicates one of three data resolution options

0 - Standard, 11 bits, plus sign

1 - High, 14 bits, plus sign

2 - Low, 8 bits, plus sign

External Synchronization

This character indicates whether external synchronization is desired.

0 - No

1 - Yes

Device Identification

This digit indicates one of two ADC's.

0 - First ADC

1 - Second ADC

I/O Area Parameter

I/O area parameter is the label of the user's 1-word input area. After the subroutine returns control to the user, this area contains the requested data. The data is in the following format.

Bits

0 - Sign

1-14 - Data bits (1-8 if low, 1-11 if standard, 1-14 if high)

15 - Overload indicator

Point Address Parameter

The point address parameter is the label of a 1-word location that contains the multiplexer address of the point to be read. This location need not immediately follow the input area. The multiplexer address has the following format:

Bits

0-2 - Not used

3 - 1 for solid state; 0 for relay

4-5 - Not used

6-15 - Multiplexer point address

Special Condition Parameter

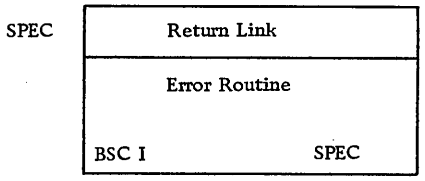
(See Basic Calling Sequence.)

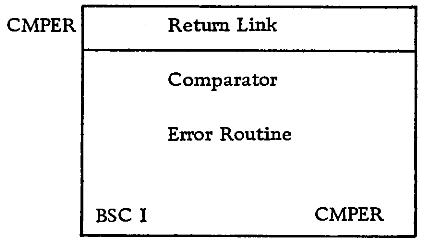
ANALOG INPUT RANDOM READ SUBROUTINE

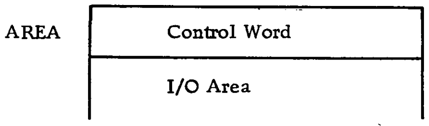
This subroutine reads in analog data for a number of random points under data channel control. The points read can be any mixture of relay and solid-state analog points; however, if the overlap feature is being used, at least 200 solid state points must be read between relay points to avoid an overlap error. The AIRN subroutine can be in core storage at the same time as the AIPTN and AISQN subroutines but cannot operate simultaneously with either. If a continuous scan is performed, both the AIRN subroutine and the data area must be in skeleton.

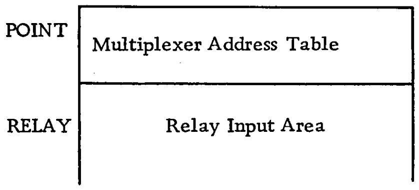
Calling Sequence

LIBF	AIRN	
DC	/xxxx	(Control)
DC	AREA	(I/O area)
DC	POINT	(Point address)
DC	SPEC	(Special Condition Parameter)
DC	CMPER	(Comparator Error Routine)
DC	RELAY	(Relay Area)





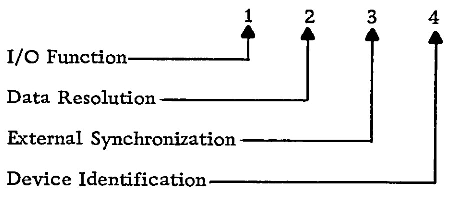




The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits, as shown below:



I/O Function

The I/O function digit specifies the operation to be performed by the AIRN subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters*
Test	0	Control
Read Without Overlap	1	Control, I/O Area, Point, Special Condition, Comparator error routines
Read Without Overlap and Without Error Retries	2	Control, I/O Area, Point, Special Condition, Comparator error routine
Read With Overlap	3	Control, I/O Area, Point, Special Condition, Comparator error routine, Relay Table
Read With Overlap and Without Error Retries	4	Control, I/O Area, Point, Special Condition, Comparator error routine, Relay Table
Reset	5	Control

*Any parameter not required for a function must be omitted.

Test. Branches to LIBF+2 if the analog feature is busy, or to LIBF+3 if the analog feature is not busy.

Read Without Multiplexer Overlap. Reads the converted values for both relay and solid-state analog points into the I/O area table(s). Table complete interrupts and comparator interrupts may occur during the operation.

Table-complete interrupts occur after the last word of a data table has been processed. (The user can suppress this interrupt by placing a 1 in bit position 1 of the I/O area control word. However, the subroutine requires this interrupt on the last table of the chain and will force the interrupt by clearing bit position 1 of the control word for the last table.) Whenever these interrupts occur (except on the last table), the subroutine branches to the address specified in the special condition parameter with a code of 65 in the A-register. The subroutine is "busy" until all of the data has been read and the last table interrupt has occurred.

A comparator interrupt occurs if the comparator feature is attached and an analog value is found to be out of the specified limits or an overload condition occurs. When this interrupt occurs, the subroutine exits to the user's special condition routine with an error code of 69 in the A-register and the comparator DSW in the Q-register. When the user's routine returns control, the AIRN subroutine exits normally, as if no error had occurred. The only effect of this error on the current operation is that no more limit checking is done until the AIRN subroutine has read the comparator status word (error indicator and multiplexer address) into the Q-register. Point conversions are not delayed.

Overload, parity, and storage protect interrupts can also occur and cause a branch via the analog input error parameter.

Read Without Multiplexer Overlap and Without Error Retries. This is the same as read without multiplexer overlap, except that on an error, no retries will be attempted. If an error occurs, AIRN will branch to the user's special condition routine with an error code of 65 in the A-register. The user is expected to return to AIRN to continue.

Read With Multiplexer Overlap. Reads converted solid-state values into the I/O area and converted relay values into the relay area. Table-complete interrupts and relay interrupts may occur indicating conversion complete conditions. The table-complete interrupt is as described above for the read without overlap function.

A relay interrupt occurs when a relay-point value has been converted and can be read into core storage. The subroutine reads the converted value into the next available word in the relay data table.

Error conditions detected are indicated by relay-misplaced interrupt, which occurs if a relay conversion is not complete when a second relay point is encountered in the point address table. When this interrupt occurs, the subroutine exits to EAC. The effect of this error on the current operation is that the second point is not converted. The equipment skips that point and continues with solid-state conversions. A comparator limit error is described in read without multiplexer overlap.

Read With Multiplexer Overlap and Without Error Retries. This is the same as read with multiplexer overlap, except that on an error, no retries will be attempted. If an error occurs, AIRN will branch via the user's special condition routine with an error code of 65 in the A-register. The user is expected to return to AIRN.

Reset. Halts all analog input operations in progress and resets all indicators.

Data Resolution

This digit indicates one of three data resolution options.

0 - Standard, 11 bits plus sign

1 - High, 14 bits plus sign

2 - Low, 8 bits plus sign

External Synchronization

This digit indicates whether external synchronization is desired.

0 - No

1 - Yes

Device Identification

This digit indicates one of two ADC's.

0 - First ADC

1 - Second ADC

I/O Area Parameter

The I/O area parameter is the label of the control word that precedes the user's I/O area. Since chaining is permitted with the AIRN subroutine, the chaining bit and the suppress interrupt indicator bit are meaningful. The chaining indicator controls chaining of both the data and multiplexer tables. If chaining is indicated, the word immediately following each data table references the next table in the data chain. The address word following each multiplexer address table references the next table in the address-table chain.

If the function is read without multiplexer overlap, the data word count is equal to the number of ADC values for the data table (equal to the number of multiplexer addresses in the corresponding multiplexer table). If the function is read with multiplexer overlap, the data word count is equal to the number of ADC values for the data table (equal to the number of solid state multiplexer addresses in the corresponding multiplexer table). Limit words are not included in either count.

The subroutine reads in 16-bit values having the following format:

Bits

0 - Sign

1-14 - Data bits (1-8 if low, 1-11 if standard, 1-14 if high)

15 - Overload indicator

Point Address Parameter

This parameter is the label of the first multiplexer address table to be used in the read operation. The subroutine reads the ADC value for each multiplexer addressed point into a data word in the corresponding data table (or relay table if the function is read with multiplexer overlap). A multiplexer address table contains address words and limit words. Limit words should be used only if the comparator feature is attached. In addition, if the overlap feature is attached, limit words should only be used with solid state multiplexer addresses.

These two types of words are interleaved in the table, with the multiplexer address word preceding its associated limit word.

Address Word Format

Bit

0 - Not used

1 - 1-limit word follows; 0-no limit word follows

2 - 1-compare limit word; 0-don't compare limit word

3 - 1-solid state; 0-relay

4-5 - Not used

6-15 - Multiplexer point address

Comparator Word Format

Bit

0 - High-limit sign

1-7 - High-limit data bits

8 - Low-limit sign

9-15 - Low-limit data bits

Special Condition Parameter

This parameter specifies the entry location of a user's special condition subroutine. When the branch to the user's routine is performed, the error code is contained in the A-register. The user's routine must return control to the AIRN subroutine after performing the desired operation.

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must be in the skeleton.

Comparator Error Routine

This parameter is the address of the user-written subroutine to which the program will branch on a comparator error.

If the I/O area and the I/O subroutine are in the system skeleton, the comparator error routine must be in the skeleton.

Relay Area Parameter

This parameter is the label of the first word into which relay multiplexer ADC values are to be read if the multiplexer overlap feature is attached. Since chaining is not permitted with relay data, this area must be large enough to contain all of the relay readings for an entire operation.

Example Without Overlap

An example of a multiplexer address table and a corresponding data table follows. This example illustrates reading in five solid-state and relay analog points, with some limit-checking.

```
POINT  +0 DC /1006 (multiplexer address for SS point #6)
       +1 DC /700A (multiplexer address for SS point #10)
       +2 DC /3504 (limit word for point #10)
       +3 DC /4002 (multiplexer address for R point #2)
       +4 DC /1001 (limit word for point #2 - not used)
       +5 DC /0030 (multiplexer address for R point #48)
       +6 DC /100C (multiplexer address for SS point #12)

AREA   +0 DC 5 (word count)
       +1 DC 0 (register for value of SS point #6)
       +2 DC 0 (register for value of SS point #10)
       +3 DC 0 (register for value of R point #2)
       +4 DC 0 (register for value of R point #48)
       +5 DC 0 (register for value of SS point #12)
```

Example With Overlap

An example of a multiplexer address table and corresponding solid-state and relay tables follows. This example illustrates the reading in of many solid-state and two relay analog points, with some limit checking.

```
POINT  +0 DC /1006 (multiplexer address for SS point #6)
       +1 DC /700A (multiplexer address for SS point #10)
       +2 DC /3504 (limit word for point #10 - used)
       +3 DC /4002 (multiplexer address for R point #2)
       +4 DC /1001 (limit word for point #2 - not used)
       +5 DC /100C (multiplexer address for SS point #12)
        *   .   .
        *   .   .
        *   .   .
        *   .   .
        *   .   .
     +317  DC /0030 (multiplexer address for R point #48)

AREA   +0 DC 3 (word count)
       +1 DC 0 (register for value of SS point #6)
       +2 DC 0 (register for value of SS point #10)
       +3 DC 0 (register for value of SS point #12)

RELAY  +0 DC 0 (register for value of R point #2)
       +1 DC 0 (register for value of R point #48)
```

*NOTE: At least 200 SS points must be scanned between R point #2 and R point #48. More than 200 SS points can be scanned if desired.

The IBM 1800 Subroutine Library contains a set of special subprograms which transfer control to certain of the analog/digital subroutines previously described in this manual. Specifically, the functions of each of these subprograms are:

1. To decode the control parameter.
2. To set up data and address tables.
3. To set up and execute the calling sequences to the proper subroutine.
4. To handle error and exception conditions.
5. To return control to the next sequential statement in the calling program.

The subprogram names and the analog/digital subroutine to which they provide linkage are listed below.

FORTRAN Subprogram	Subroutine Called
AIP	AIPTN (Analog Input Single Read)
AIS	AISQN (Analog Input Sequential)
AIR	AIRN (Analog Input Random)
CO	DAOP (Digital Analog Output)
DAC	DAOP
DO	DAOP
CS	DINP (Digital Input)
CSC	DICMP (Digital Input Read and Compare)
CSX	DIEXP (Digital Input and Expand)
VS	DINP (Digital Input)
VSC	DICMP (Digital Input Read and Compare)
VSX	DIEXP
PI	DINP
PIC	DICMP
PIX	DIEXP
DI	DINP
DIC	DICMP
DIX	DIEXP
PO	DAOP

NOTE: To fully understand these FORTRAN subprograms, the user must be familiar with the descriptions of the corresponding IOCS subroutines.

CALL STATEMENTS

The following paragraphs describe the CALL statements used to gain access to the FORTRAN subprograms. For detailed descriptions of the parameter, refer to the appropriate subroutine.

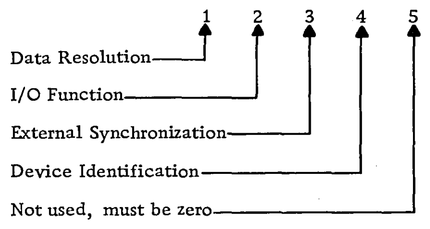
ANALOG INPUT SINGLE READ

The FORTRAN CALL statement shown below is used to call the AIPTN (analog input single read) subroutine via a FORTRAN subprogram.

```
CALL AIP (I, J, K)
```

I Parameter

The I parameter is an integer constant or integer variable that specifies various control options. It consists of 5 decimal digits for the options desired.



Data Resolution

This digit indicates one of three data-resolution options.

0 - Standard, 11 bits plus sign
1 - High, 14 bits plus sign
2 - Low, 8 bits plus sign.

I/O Function

The I/O function digit specifies the function to be performed by the AIPT subroutine. The functions, associated digital values and required parameters are listed and described below.

Function	Digital Value	Required Parameters
Test	0	I, J
Read	1	I, J, K
Halt	3	I

Test. The test function determines if the analog input device called for is busy. If the device is busy, the second parameter, J (a single integer variable) will be loaded with a 1; if it is not busy, J will be loaded with a 2. The device code is used with the test function to specify the first or second ADC.

Example:

```
 5          CALL AIP (0, JTEST)
            GO TO (5, 10) JTEST
10            .
              .
              .
              .
```

In the above example, the first parameter of the CALL statement indicates the test function. The second parameter is loaded with 1 or 2 for busy or not busy, respectively. The Computed GO TO statement will transfer control back to statement 5 if the first analog input device is busy or to statement 10 if it is not busy.

NOTE: The test function can test the busy or not busy condition of any of the analog or digital I/O devices called by the FORTRAN subprograms. When an I/O subroutine is called and the device is busy, the new operation must wait until the operation already in progress is completed.

Read. Reads one relay or solid state point under direct program control.

Halt. Terminates input on the addressed device.

External Synchronization

This digit indicates whether external synchronization is desired.

0 - No
1 - Yes

Device Identification

This digit indicates one of two ADC's

0 - First ADC
1 - Second ADC

J Parameter

The J parameter is an integer variable that specifies the input word where the value of the point is to be placed.

K Parameter

The K parameter is an integer constant or integer variable that specifies the multiplexer point to be read. This variable must include a 1-bit in binary position 3 if the point to be read is a solid state point (as opposed to a relay point). The bit will be placed in position 3 automatically if the K parameter is an expression with 4096 added to the point address.

Example:

CALL AIP (21100, INWD, 75)

In this example, the first parameter specifies that a read operation, with a resolution of 8 bits and controlled by some external synchronization, is to be performed, using the first ADC. The second parameter (INWD) specifies the input word for the analog input, and the third parameter specifies that multiplexer point 75 is to be read. This is a relay point; the same numbered solid-state point would be written as 75 + 4096.

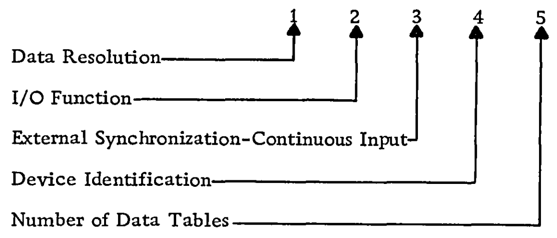
ANALOG INPUT SEQUENTIAL

The FORTRAN CALL statement shown below is used to call the AISQN (Analog Input Sequential Read) subroutine via a FORTRAN subprogram.

CALL AIS (I, J, M)

I Parameter

The I parameter is an integer constant or integer variable that specifies the control options. It consists of 5 decimal digits for the options desired.



Data Resolution

This digit indicates one of three data-resolution options.

0 - Standard, 11 bits, plus sign
1 - High, 14 bits, plus sign
2 - Low, 8 bits, plus sign

I/O Function

The I/O function digit specifies the function to be performed by the AISQN subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters
Test	0	I, J *
Read-Direct Program Control	1	I, J
Read-Data Channel Control	2	I, J
Halt this AI Data Channel	4	I
Read and Transfer (Uses Direct Program Control)	5	I, J, M

*The J parameter for the test function is a single integer variable (same as described under AIP).

Test. Tests for device busy are the same as described for the analog input single read subroutine (CALL AIP).

Read-Direct Program Control. Under direct program control, reads the specified number of analog input values into the I/O area.

Read-Data Channel Control. Under data channel control, reads the specified number of analog input values into the I/O area. If continuous I/O (see below) is specified, the subroutine reads continuously by chaining back to the beginning of the I/O area. This operation, once initiated, will continue indefinitely. For this reason, the I/O area must not be overlaid.

Halt. Terminates input on the addressed device.

Read and Transfer. Under direct program control, reads the specified number of analog input values into the I/O area. After each point is read, the routine transfers to the EXTERNAL declared subprogram specified by M. This subprogram, written by the user, must use two parameters: the address of the point read and the address of the variable in the I/O area filled by the reading, in that order.

External Synchronization - Continuous Input

This digit indicates whether external synchronization and/or continuous input operation are desired.

0 - No external synchronization; no continuous input
1 - External synchronization; no continuous input
2 - No external synchronization; continous input (data channel only)
3 - External synchronization; continuous input (data channel only)

Device Identification

This digit indicates one of two ADCs.

0 - First ADC
1 - Second ADC

Number of Data Tables

This digit specifies the number of sets of variables (three variables per set) that identify I/O area boundaries. Up to nine sets of variables (nine I/O areas) can be specified when data channel control is also specified. Only one I/O area can be used if direct program control is specified. This digit must correspond to the actual number of sets included in the J parameter.

J Parameter

The J parameter consists of one or more sets of variables (three variables per set) that define the boundaries of the data tables to be used and the starting multiplexer addresses. The number of J parameter sets is defined by the fifth digit of the I parameter.

There must be one set of J parameters for each data table. The first variable of the J parameter set must be the last variable of the data table, the second must be the first variable of the data table, and the third must be the starting multiplexer address. When only one table is required, the DIMENSION statement must allow two extra variable positions in the data table for the word count and the starting multiplexer address. These two words are set up by the AIS routine when it is called.

NOTE: Arrays are stored in a sequence opposite to the sequence of I/O data transmission. Therefore, if the data table is defined as an array, data is first transmitted to or from the variable with the largest subscript.

If the data table is to be "chained", one additional set of J parameters will be required for each data table in the chain. In addition, the DIMENSION statement must allow three extra variable positions in the first data table. The first two extra positions are the same as for a single table, and the third extra position is for the chaining address, which is also placed in the table area by the AIS routine. The DIMENSION statement must allow four extra variable positions in the second data table. The first extra position in the second data table is the channel address register (CAR) and corresponds to the chaining address in the first data table. This address is placed in the table area by the AIS routine. The second and third extra positions are the word count and starting multiplexer address of the second data table, respectively. The fourth extra position is the chaining address to the next table in the chain.

Each additional table, not including the last table, must allow these four extra variable positions. In the last table, only three extra variable positions are needed because the chaining address is omitted.

If the continuous scan mode is to be used, all tables will require four extra variable positions. The chaining address of the last data table in the chain will be set to the address of the CAR check word in the first data table.

Relay point addresses must include a zero in bit position three to distinguish them from solid-state point addresses. Solid state point addresses must have a one in bit position three. In addition, relay points and solid state points cannot be intermixed with a table; however, they can be interleaved by chaining the tables together.

M Parameter

The M parameter consists of the name of an EXTERNAL subprogram written by the user to process each input reading as it is obtained when using the read and transfer function. This subprogram must be written to use two parameters: the address of the point read and the address of the variable in the I/O area filled by the readings.

Example 1: (Figure 14)

```
        DIMENSION              INDAT (27)
            .
            .
            .
            .
            .
        CALL AIS (01011, INDAT (1), INDAT (27), 81)
```

The first parameter, I, specifies:

0 - Standard data precision.
1 - Read-direct program control.
0 - No external synchronization; no continuous input.
1 - Second ADC.
1 - One data table.

The second parameter, J, specifies:

INDAT (1)	Last variable position of the data table, where the last multiplexer point reading will be placed.
INDAT (27)	First variable position of the data table, where the data table word count is placed.
81	Multiplexer address is placed in INDAT (26).

The DIMENSION INDAT (27) specifies that the data table will be 27 variables in length, but the first data point read will be placed in the position of the data table that would correspond to INDAT (25).

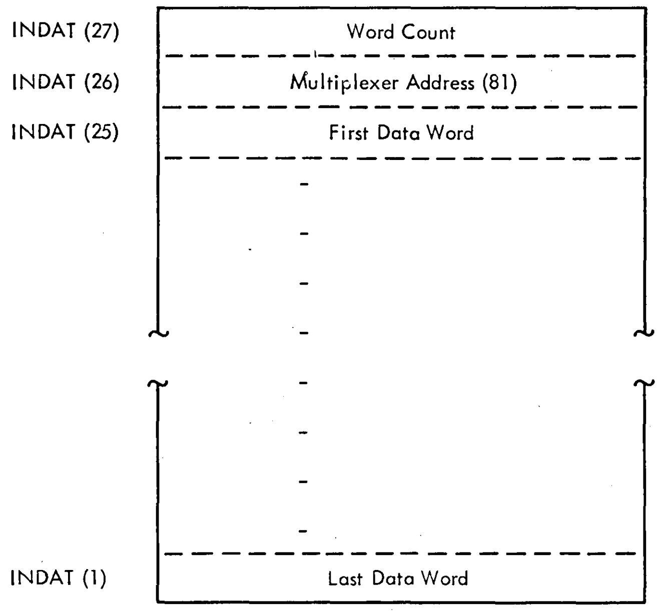


Figure 14. Single Multiplexer Data Table

Example 2:

```
DIMENSION              IN1 (21), IN2 (14)
    .
    .
    .
    .
    .
CALL AIS (12002, IN1 (1), IN1 (21), 5, IN2 (1), IN2 (14), 17)
```

The first parameter, I, specifies:

1 - 14 bit resolution.
2 - Read-data channel control.
0 - No external synchronization; no continuous input.
0 - First ADC
2 - Two data tables

The second parameter, J, specifies

IN1 (1) Last variable position of the first data table, where the chaining address is placed.
IN1 (21) First variable position of the first data table, where the word count of the first table is placed.
5 Multiplexer address is placed in IN1 (20).
IN2 (1) Last variable position of the second data table, where the last Multiplexer point reading will be placed.
IN2 (14) First variable position of the second data table, where the chaining address from the first table is placed.
17 Multiplexer address is placed in IN2 (12).

The DIMENSION IN1 (21), IN2 (14) specifies that the first data table will be 21 variables in length and the second data table will be 14 variables in length. The first data point read will be placed in the position of the first data table that corresponds to IN1 (19) and the last data point read will be placed in the position of the second data table that corresponds to IN2 (1).

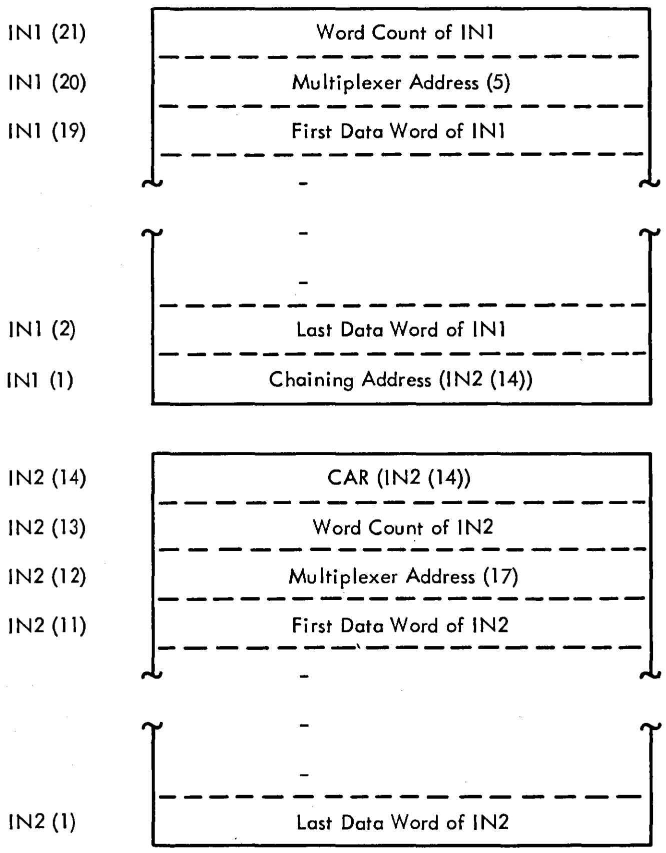


Figure 15. Example of Two Chained Multiplexer-Data Tables

Example 3:

```
     DIMENSION              IN1 (23), IN2 (23), IN3 (23)
         .
         .
         .
         .
         .
     CALL AIS (12203, IN1 (1), IN1 (23), 1, IN2 (1), IN2 (23), 20,
               IN3 (1), IN3 (23), 40)
```

The first parameter, I, specifies:

1 - 14 bit resolution.
2 - Read-data channel control.
2 - No external synchronization, continuous input.
0 - First ADC.
3 - Three data tables.

The second parameter, J, specifies:

IN1 (1) Last variable position of the first data table, where the chaining address of the second data table is placed.
IN1 (23) First variable position of the first data table, where the chaining address from the third data table is placed.
1 Multiplexer address is placed in IN1 (21).
IN2 (1) Last variable position of the second data table, where the chaining address of the third data table is placed.
IN2 (23) First variable position of the second data table, where the chaining address from the first data table is placed.
20 Multiplexer address is placed in IN2 (21).
IN3 (1) Last variable position of the third data table, where the chaining address of the first data table is placed.
IN3 (23) First variable position of the third data table, where the chaining address from the second data table is placed.
40 Multiplexer address is placed in IN3 (21).

The DIMENSION IN1 (23), IN2 (23), IN3 (23) specifies that the first data table will be 23 variables in length, the second data table will be 23 variables in length, and the third data table will be 23 variables in length. The first data point read will be placed in the position of the first data table that corresponds to IN1 (20) and the last data point read will be placed in the position of the third data table that corresponds to IN3 (2).

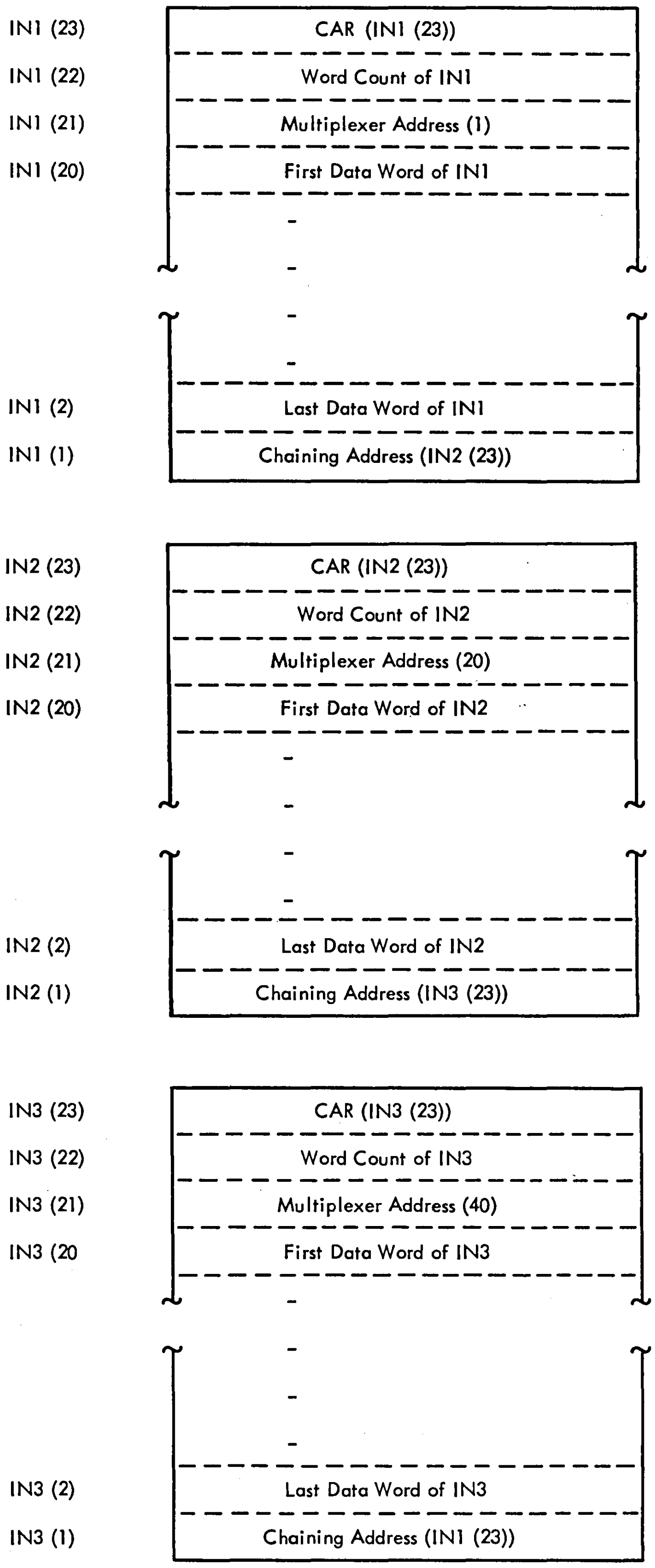


Figure 16. Example of Continuous Input Multiplexer-Data Tables

ANALOG INPUT RANDOM READ

The FORTRAN CALL statement shown below is used to call the AIRN subroutine.

```
CALL AIR (I, J, K, N, M)
```

I Parameter

The I parameter is the same as that described for analog input sequential, except the I/O function is different and that external synchronization is not allowed with an overlap request. The I/O function digit specifies the function to be performed by the AIRN subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Function	Required Parameters
Test	0	I, J*
Read-Without Multiplexer Overlap With Comparator	1	I, J, K, N, M
Read-Without Multiplexer Overlap Without Comparator	2	I, J, K, N
Reset	4	I
Read-With Multiplexer Overlap With Comparator	5	I, J, K, N, M
Read-With Multiplexer Overlap Without Comparator	6	I, J, K, N

*The J parameter for the test function is a single integer variable (same as described under AIP).

Test. Tests for device busy are the same as described for the analog input-single read subroutine (see AIP).

Read-Without Multiplexer Overlap With Comparator. Reads the solid state and relay points, specified in the multiplexer address table, into the successive words of the data table. Each value is compared with the limits for the point read when specified by the control bits in the address word. Each point is read, and if specified, compared before the next point is used, and a transfer to the named subprogram in the M parameter is made if a limit is violated.

Read-Without Multiplexer Overlap Without Comparator. This function is the same as described for read-without multiplexer overlap with comparator, except the comparator is not used, therefore parameter (M) that names the violation subprogram must be omitted when this function is specified.

Reset. Halts all I/O operations of this analog function.

Read-With Multiplexer Overlap With Comparator. Reads the solid state points, specified in the multiplexer address table, into successive words of the data table for solid state points. Each value is compared with the limits of the point read, when specified by the control bits in the address word. Also reads relay points, included in the multiplexer address table, into successive words of the data table for relay points. Conversion of the relay point does not stop the conversion of the solid state points that follow the relay point in the multiplexer address table. Each relay point requires 10, 000 μs for conversion, therefore, 200 solid state points must be converted between each relay point. If a limit is violated, a transfer is made to the named subprogram in the M parameter.

Read-With Multiplexer Overlap Without Comparator. This function is the same as described for read-with multiplexer overlap with comparator, except the comparator is not used, therefore parameter (M) that names the violation subprogram must be omitted when this function is specified.

J Parameter

The J parameter is the same as described for the analog input sequential subroutine, except that the multiplexer point variable is not included in the set. Therefore, the data tables for AIR require one less extra variable than for AIS. The multiplexer point addresses are specified in the K parameter.

K Parameter

The K parameter consists of one or more pairs of variables which define the boundaries of the multiplexer address tables and the limit words that correspond to the multiplexer addresses. The number of K parameter pairs (specifying multiplexer address tables) must be the same as the number of J parameter pairs. One extra variable position must be included in the DIMENSION statement for the K parameter if, this is the first multiplexer address table in a chain (the extra variable position will be used for the chaining address to the next table), or this is the last multiplexer address table in a chain (the extra variable position will be used to store the chaining address of the previous table in CAR). Two extra variable positions must be provided if this is an intermediate multiplexer address table in a chain (the first extra variable position will be used to store the chaining address of the previous table in CAR and the second will be used for the chaining address to the next table in the chain). If continuous input is specified, each table (including the first) must include two extra variable positions. The first extra variable will be used to store (in CAR) the chaining address from the last table; the second extra variable will be used for the chaining address to the next table. The multiplexer addresses and limit words must be placed in the tables defined by the K parameter. This can be done by using the FORTRAN DATA statement.

N Parameter

The N parameter specifies the first core storage address of a distinct table where relay points will be stored when using the multiplexer overlap feature. This could be the highest subscripted element of an array. The N parameter must always be included even if no overlap is being used (i.e., CALL AIR (I, J, M) is illegal); however, if no overlap is used, the dummy address specified by N will not be used by the AIRN subroutine). Relay points will be stored in the same data table used for solid state points when there is no overlap feature on the machine. Each point will be read into the table before the next point reading is initiated. Since chaining is not permitted with relay data tables, the data table must be large enough to contain all the relay readings for an entire operation.

M Parameter

The M parameter must always be included when the comparator is used, it specifies the name of the subprogram to which the AIR subprogram will branch if a limit violation is detected. The M parameter must be declared in an EXTERNAL statement with the same core load that calls AIR.

The subprogram must be written to include two parameters. The first will be the address of a type-of-limit indicator. If the value is one, a low-limit violation has occurred, if the value is two, a high-limit violation has occurred, or if the value is three, an overloaded point caused the violation. The second parameter will be the address within the AIR routine where the number of the input point is stored.

Example 1:

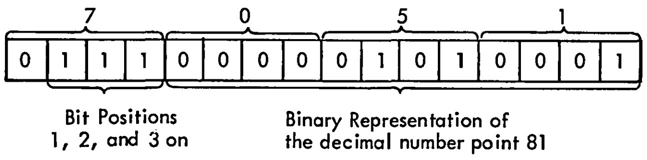
```
EXTERNAL                         LIM
DIMENSION                        NAREA (4), IM (6)
DATA IM (6), IM (5)/Z7051, Z3C1E/
DATA IM (4), IM (3)/Z7056, Z3214/
DATA IM (2), IM (1)/Z7025, Z4628/
CALL AIR (01011, NAREA(1), NAREA(4), IM(1),
IM(6), 0, LIM)
```

Including the DATA statement results in the following:

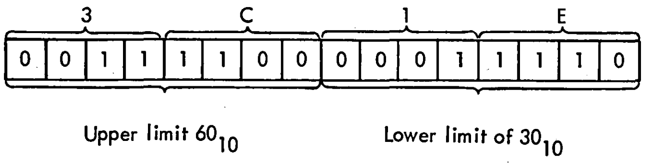
IM(6) = Point 81 with bits in positions one, two, and three.
IM(5) = Limits 60 - 30 in left and right bytes, respectively.
IM(4) = Point 86 with bits in positions one, two, and three.
IM(3) = Limits 50 - 20 in left and right bytes, respectively.
IM(2) = Point 37 with bits in positions one, two, and three.
IM(1) = Limits 70 - 40 in left and right bytes, respectively.

The derivation of the hexadecimal constants for IM(6) and IM(5) is as follows:

IM(6)


7
0
5
1
0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 1
Bit Positions 1, 2, and 3 on
Binary Representation of the decimal number point 81

IM(5)



The characters 7, 0, 5, 1 and 3, C, 1, E are the hexadecimal representation for each set of four bits in the table words. These are the values to be used in the DATA statement preceded by a letter Z.

(The advantage of the DATA statement over the method of table generation as shown in Example 2 is that the DATA statement does not generate any object time executable code and thus requires no object execution time. The FORTRAN compiler generates the table as part of the object program from the information supplied in the DATA statement.)

In this example, .the I parameter (01011) specifies:

0 - Standard data resolution
1 - Read-without multiplexer overlap with comparator
0 - No external synchronization; no continuous input
1 - Second ADC
1 - One data table

The J parameter (NAREA(1), NAREA(4)) specifies:

NAREA(1) The last data point to be read, compared, and entered into the data table (multiplexer address 37).
NAREA(4) The word count (3) of the data table NAREA.

The K parameter (IM(1), IM(6)) specifies:

IM(1) The last specified limit word, which will be the last entry in the multiplexer address table.
IM(6) The first multiplexer address to be read and compared, and the first entry in the multiplexer address table.

The multiplexer addresses of the K parameter (in this example, table variables with even numbered subscripts) can have 1-bits placed in positions one, two, and three of the address word, which can mean:

MULTIPLEXER ADDRESS TABLE

IM(6)	Multiplexer address 81
IM(5)	Limit word 30-66 60-30
IM(4)	Multiplexer address 86
IM(3)	Limit word 20-50 50 -20
IM(2)	Multiplexer address 37
IM(1)	Limit word 40-70 70 -40

DATA TABLE

NAREA(4)	Word count (3) of NAREA
NAREA(3)	First data word (point 81)
NAREA(2)	Second data word (point 86)
NAREA(1)	Last data word (point 37)

Figure 17. Single Multiplexer Address and Data Tables

1. One-bits in all three positions means a limit word follows, the comparator feature is used, and this is a solid-state point.
   If no bit is in position three, a relay point is specified.
2. One-bits in positions one and three means a limit word follows, no comparator feature is used, and this is a solid-state point.
   If no bit is in position three, a relay point
3. A one-bit in position three only means no limit word follows, no comparator feature is used, and this is a solid-state point. If no bit is in position three, a relay point is specified.

NOTE: The combination of 1-bits in positions two and three only is not a logical condition since this would mean no limit word follows, the comparator feature is used, and this is a solid state point, respectively. Both the low and high limit values are placed in one word. To accomplish this, the hexadecimal specification in the DATA statement can be used as shown in Example 1 or the high limit can be multiplied by 256 as shown in Example 2.

The N parameter (0) specifies:

0 Since no multiplexer overlap will take place, both solid state and relay points can be read under this condition, but each point will be converted before the next point is read. If no multiplexer overlap is used (as in this example) a zero must be included in the place of the N parameter.

In this example, the M parameter (LIM) specifies:

LIM A user written subprogram that the AIR subprogram will branch to if a limit violation occurs. The parameters of this subprogram might be LH and MPT, where LH would contain the address of the violation indication, and MPT would contain the number of the multiplexer point that violated the limits. Both LH and MPT are parameters of LIM, so the violation subprogram can reference them. A Computed GO TO statement could have been used to distinguish the low limit, high limit, and overload violations.

NOTE: The following examples are to clarify the J, K, N parameters, by giving examples of the possible variations of chaining. Therefore, each example will use the same M parameter described for example 1.

Example 2: (Figure 18)

```
EXTERNAL     LIM
DIMENSION    NAREA (202), NTABL (202),
             IM (403), IK (403), F007(2)

IM(403) =  6 * 4096 + 3         Relay point 3
IM(402) = 10 + 40 * 256         Limits 10-40
IM(401) = 7 * 4096 + 1          Point 1
IM(400) = 15 + 35 * 256         Limits 15-35
IM(399) = 7 * 4096 + 81         Point 81
IM(398) = 15 + 35 * 256         Limits 15-35
      .
      .
SEE NOTE BELOW
      .
      .
IM(3) = 7 * 4096 + 86           Point 86
IM(2) = 15 + 35 * 256           Limits 15-35

IK(402) = 6 * 4096 + 7          Relay point 7
IK(401) = 50 + 80 * 256         Limits 50-80
IK(400) = 7 * 4096 + 37         Point 37
IK(399) = 15 + 35 * 256         Limits 15-35
IK(398) = 7 * 4096 + 44         Point 44
IK(397) = 15 + 35 * 256         Limits 15-35
      .
      .
SEE NOTE BELOW
      .
      .
IK(2) = 7 * 4096 + 77           Point 77
IK(1) = 15 + 35 * 256           Limits 15-35

CALL AIR (05012, NAREA (1), NAREA (202), NTABL (1),
NTABL (202), IM (1), IM (403), IK (1), IK (403), F007 (2), LIM)
```

NOTE: Assume that at least 200 solid state points have been specified between the two relay points 3 and 7.

In this example the I parameter (05012) specifies:

0 - Standard data resolution
5 - Read-with multiplexer overlap with comparator
0 - No external synchronization; no continuous input
1 - Second ADC
2 - Two data tables

The J parameter (NAREA(1), NAREA(202), NTABL(1), NTABL(202)) specifies:

NAREA(1) The chaining address to the data table NTABL(202), which is the last entry in this data table.
NAREA(202) The word count of the data table NAREA.
NTABL(1) The last read, compared, and stored data point of the entire chain, multiplexer address 77.
NTABL(202) The chaining address of NAREA(1), the previous data table, stored in CAR as (NTABL (202)).

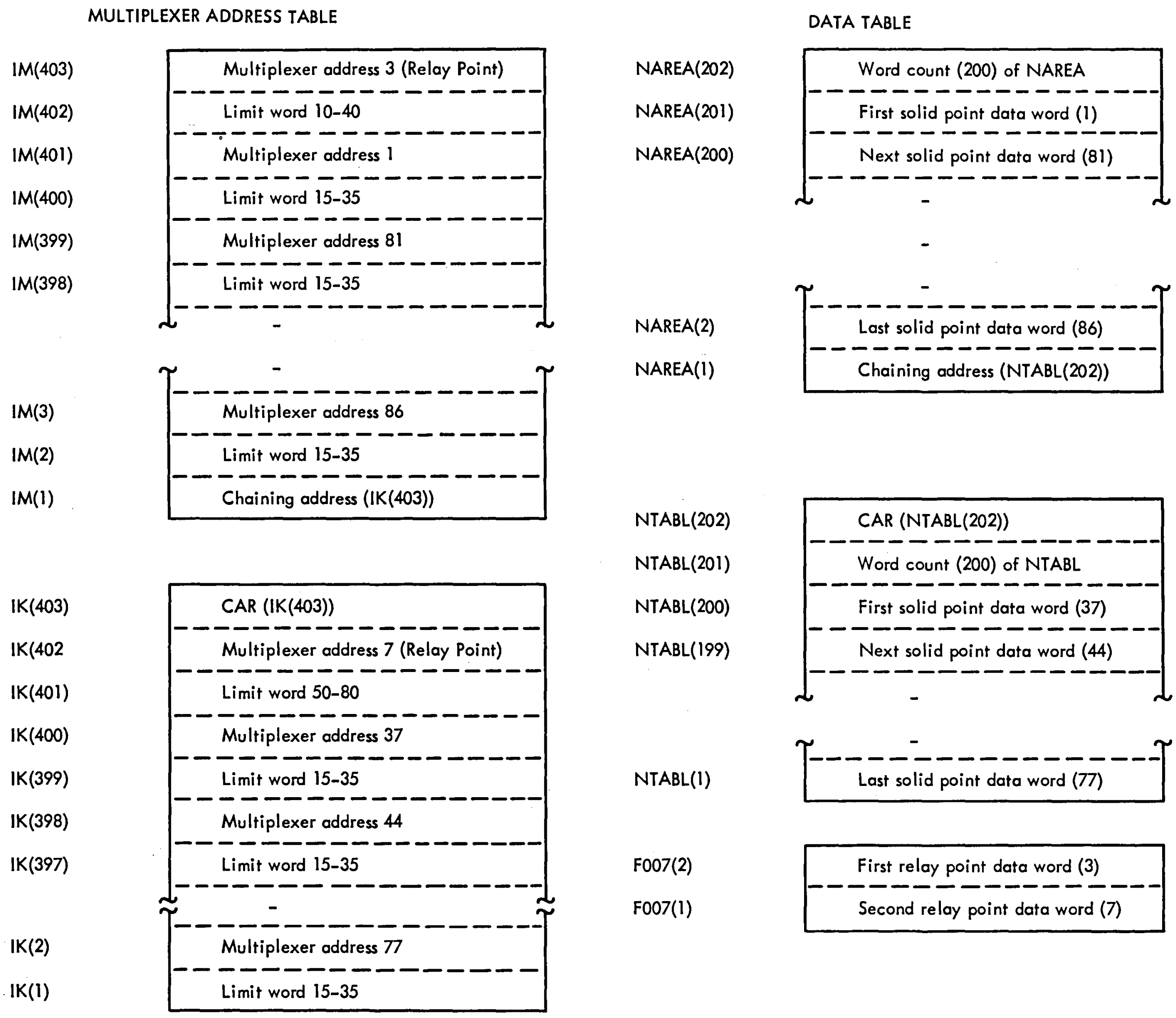
The K parameter (IM(1), IM(403), IK(1), IK(403)) specifies:

IM(1) The chaining address to the multiplexer address table IK(403), which is the last entry in this multiplexer address table.
IM(403) The first multiplexer address (relay point 3) of the chain to be read and compared, and the first entry in this multiplexer address table.
IK(1) The last specified limit word (limit word for point 77) of the chain, which is the last entry in this multiplexer address table.
IK(403) The chaining address of IM(1), the previous multiplexer address table, stored in CAR as (IK(403)).

In this example, the N parameter (IF007(2)) specifies:

IF007(2) The core storage address where the relay data table will start. The address will automatically be incremented by one each time an entry is made in the relay data table.

The M parameter (LIM) is the same as described for Example 1.



MULTIPLEXER ADDRESS TABLE
IM(403) Multiplexer address 3 (Relay Point)
IM(402) Limit word 10-40
IM(401) Multiplexer address 1
IM(400) Limit word 15-35
IM(399) Multiplexer address 81
IM(398) Limit word 15-35
IM(3) Multiplexer address 86
IM(2) Limit word 15-35
IM(1) Chaining address (IK(403))
IK(403) CAR (IK(403))
IK(402 Multiplexer address 7 (Relay Point)
IK(401) Limit word 50-80
IK(400) Multiplexer address 37
IK(399) Limit word 15-35
IK(398) Multiplexer address 44
IK(397) Limit word 15-35
IK(2) Multiplexer address 77
IK(1) Limit word 15-35
DATA TABLE
NAREA(202) Word count (200) of NAREA
NAREA(201) First solid point data word (1)
NAREA(200) Next solid point data word (81)
NAREA(2) Last solid point data word (86)
NAREA(1) Chaining address (NTABL(202))
NTABL(202) CAR (NTABL(202))
NTABL(201) Word count (200) of NTABL
NTABL(200) First solid point data word (37)
NTABL(199) Next solid point data word (44)
NTABL(1) Last solid point data word (77)
F007(2) First relay point data word (3)
F007(1) Second relay point data word (7)

Figure 18. Example of Two Chained Multiplexer Address and Data Tables Using the Multiplexer Overlap Feature

Example 3: (Figure 19)

EXTERNAL	LIM
DIMENSION	NAREA (5), NTABL (5), IM (7)
.	
.	
IM(6) = 7*4096 + 81	Point 81
IM(5) = 15 + 35*256	Limits 15-35
IM(4) = 3	Relay point 3
IM(3) = 7*4096 +86	Point 86
IM(2) = 15 + 35*256	Limits 15-35

```
CALL AIR (01012, NAREA (1), NAREA (5), NTABL (1),
NTABL (5), IM (1), IM (6), IM (2), IM (7), 0, LIM)
```

In this example, the I parameter (01012) specifies:

0 - Standard data resolution
1 - Read-without multiplexer overlap with comparator
0 - No external synchronization; no continuous input
1 - Second ADC
2 - Two data tables

The J parameter (NAREA(1), NAREA(5), NTABL(1), NTABL(5)) specifies:

NAREA(1) The chaining address to NTABL(5), which is the last entry in this data table.
NAREA(5) The word count of the data table NAREA.

MULTIPLEXER ADDRESS TABLE

IM(7)	CAR chaining address (IM(8))
IM(6)	Multiplexer address 81
IM(5)	Limit word 15-35
IM(4)	Multiplexer address 3
IM(3)	Multiplexer address 86
IM(2)	Limit word 15-35
IM(1)	Chaining address (IM(8))

DATA TABLES

NAREA(5)	Word count (3) of NAREA
NAREA(4)	First data word (point 81)
NAREA(3)	Second data word (point 3)
NAREA(2)	Last data word (point 86)
NAREA(1)	Chaining address (NTABL(5))

NTABL(5)	CAR (NTABL(5))
NTABL(4)	Word count (3) of NTABL
NTABL(3)	First data word (point 81)
NTABL(2)	Second data word (point 3)
NTABL(1)	Last data word (point 86)

Figure 19. Example of Two Chained Data Tables and Multiplexer Address Table Chained to Itself

NTABL(1) The last read, compared, and stored data point of the entire chain, multiplexer address 86.
NTABL(5) The chaining address of NAREA(1), the previous data table, stored in CAR as (NTABL(5)).

The K parameter (IM(1), IM(6), IM(2), IM(7)) specifies:

IM(1) The chaining address this multiplexer address table's CAR (IM(7)), the last entry in this multiplexer address table.
IM(6) The first multiplexer address to be read and compared.
IM(2) The last specified limit word (limit word for point 86) of this multiplexer address table.
IM(7) The chaining address from IM(1), stored in CAR as (IM(7)).

Note that IM(4) contains a multiplexer point for which no limit check is required. In this case, the next word is a multiplexer point. The N parameter (0) is zero as ~~required with~~ a dummy address since there is no multiplexer overlap.

The M parameter is the same as described in Example 1.

Example 4: (Figure 20)

```
EXTERNAL                 LIM
DIMENSION                IN1 (23), IN2 (23), IN3 (23), IM (42)
    .
    .
IM(41) = 7*4096 +77      Point 77
IM(40) = 15 + 35*256     Limits 15-35
IM(39) = 7*4096 + 81     Point 81
IM(38) = 15 + 35*256     Limits 15-35
IM(37) = 7*4096 + 86     Point 86
IM(36) = 15 + 35*256     Limits 15-35
    .
    .
    .
    .
IM(3) = 7*4096 + 44      Point 44
IM(2) = 15 + 35*256      Limits 15-35

CALL AIR (11203, IN1 (1), IN1 (23), IN2 (1), IN2 (23),
IN3 (1), IN3 (23), IM (1), IM (~~41~~), IM (1), IM (42),
IM (1), IM (42), 0,LIM)        42
```

In this example, the I parameter (11203) specifies:

1 - 14-bit resolution
1 - Read-without multiplexer overlap with comparator
2 - No external synchronization; continuous input
0 - First ADC
3 - Three data tables

The J parameter (IN1 (1), IN1 (23), IN2 (1), IN2 (23) IN3 (1), IN3 (23)) specifies:

IN1 (1) The chaining address to IN2 (23), which is the last entry in this data table.
IN1 (23) The chaining address from IN3 (1), the previous data table with continuous input, stored in CAR as (IN1 (23)).
IN2 (1) The chaining address to IN3 (23), which is the last entry in this data table.
IN2 (23) The chaining address from IN1 (1), the previous data table, stored in CAR as (IN2 (23)).
IN3 (1) The chaining address to IN1 (23), which is the last entry in this data table.
IN3 (23) The chaining address from IN2 (1), the previous data table, stored in CAR as (IN3 (23)).

The K parameter (IM (1), IM(42), IM (1), IM (42), IM (1), IM (42)) specifies:

IM (1) The chaining address to this multiplexer address table's CAR (IM(42)), the last entry in this multiplexer address table.
IM (41) The first multiplexer address to be read and compared.
IM (42) The chaining address from IM (1), stored in CAR as (IM (42)).

The N parameter (0) is zero as a dummy address since there is no multiplexer overlap.

The M parameter (LIM) is the same as described in Example 1.

Example 4 illustrates that continuous input requires only one multiplexer address table, since the table is merely chained to itself. However, as many multiplexer address tables as required can be used, as long as the chaining address (last variable position in the table) of the last table is the address of CAR in the first table.

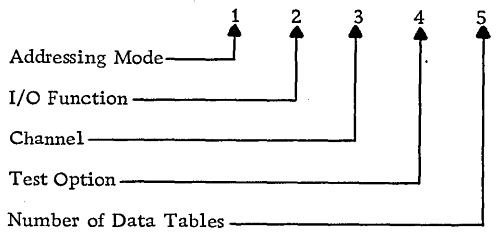
Also, in all four examples, the comparator feature has been used to illustrate the maximum relative size of the multiplexer address tables as compared with the data tables. If the comparator feature is not installed, limit words must be omitted from the multiplexer address table. The limit words must also be omitted for any multiplexer address that does not contain a bit in position one (the bit indicating a limit word follows), even if the comparator feature is installed.

DIGITAL/ANALOG OUTPUT

The FORTRAN CALL statements shown below are used to call the DAOP digital analog output subroutines for the purpose of executing contact operate, digital to analog conversion, and digital output operations.

CALL CO (I, J)	Contact Operate
CALL DAC (I, J)	Digital to Analog Conversion
CALL DO (I, J)	Digital Output
CALL PO (I, J)	Pulse Output

I Parameter. The I parameter is an integer constant or integer variable that specifies various control options. It consists of 5 decimal digits for all functions. Note that for the Halt function, I can be 04000 to terminate output.



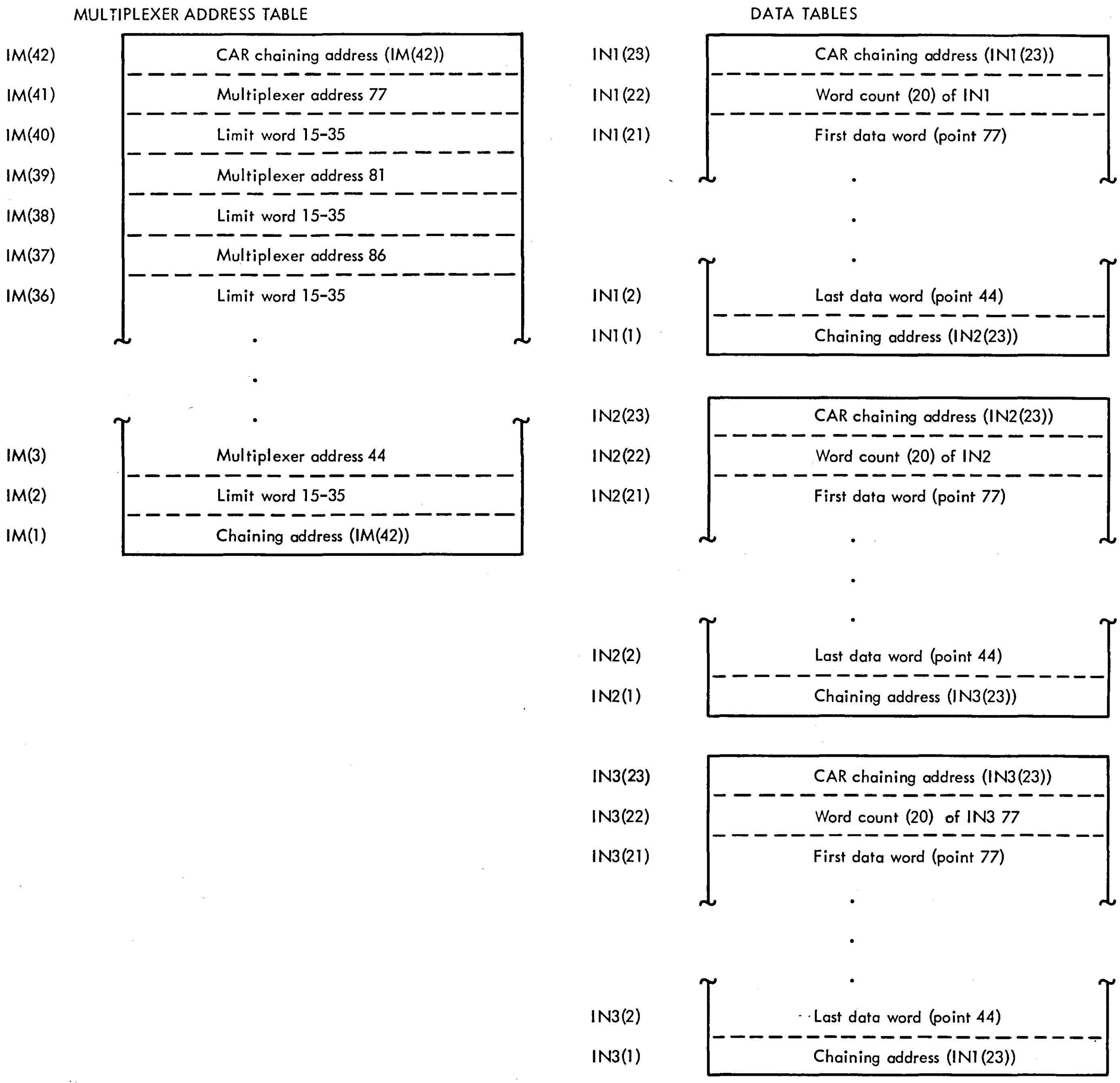

MULTIPLEXER ADDRESS TABLE
IM(42) CAR chaining address (IM(42))
IM(41) Multiplexer address 77
IM(40) Limit word 15-35
IM(39) Multiplexer address 81
IM(38) Limit word 15-35
IM(37) Multiplexer address 86
IM(36) Limit word 15-35
IM(3) Multiplexer address 44
IM(2) Limit word 15-35
IM(1) Chaining address (IM(42))
DATA TABLES
IN1(23) CAR chaining address (IN1(23))
IN1(22) Word count (20) of IN1
IN1(21) First data word (point 77)
IN1(2) Last data word (point 44)
IN1(1) Chaining address (IN2(23))
IN2(23) CAR chaining address (IN2(23))
IN2(22) Word count (20) of IN2
IN2(21) First data word (point 77)
IN2(2) Last data word (point 44)
IN2(1) Chaining address (IN3(23))
IN3(23) CAR chaining address (IN3(23))
IN3(22) Word count (20) of IN3 77
IN3(21) First data word (point 77)
IN3(2) Last data word (point 44)
IN3(1) Chaining address (IN1(23))

Figure 20. Example of Continuous Input Multiplexer Address and Data Tables

Addressing Mode

This addressing mode digit specifies one of four available addressing options: See update

0 - Random addressing
1 - Single addressing
2 - Random addressing with external synchronization
3 - Single addressing with external synchronization

Random Addressing. One multiplexer address for each output value in a random mode, must precede its associated data output value.

Single Addressing. A single address, preceding a series of data output values, can be specified.

Random Addressing With External Synchronization. Same as random addressing, except the output is externally synchronized.

Single Addressing With External Synchronization. Same as single addressing, except the output is externally synchronized.

I/O Function

The I/O function digit specifies the function to be performed by the DAOP subroutine. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters
Test	0	I, J*
Write	1	I, J
Write Pulse	2	I, J
Write Buffered	3	I, J
Halt	4	I

*The J parameter for the test function is a single integer variable (same as described under AIP).

Test. The test function is used in conjunction with the test option to determine program busy status or pulse timer busy status. The J parameter is the same as described for the analog input single read subprogram (see AIP).

Write, Write Pulse, and Write Buffered. Writes the requested number of digital/analog values, using the mode requested.

Halt. Terminates output on the addressed device.

Channel

This digit specifies the mode of data transfer for the operation.

0 - Direct program control
1 - Data channel control

Test Option

This digit tests the status of the previous operation or the pulse output timer. This digit must be zero for analog output.

0 - Test the status of the previous digital/analog output operation.
1 - Test the status of the pulse output timer.

Number of Data Tables

Same as described for analog input sequential subprogram.

J Parameter

The J parameter consists of one or more pairs of integer variables that define the boundaries of the data tables (or combined data and point address tables) to be used.

There must be one pair of J parameters for each data table. The first and second variables of the J parameter set must specify the last and first variables of the data table, respectively. If random addressing is used, each point address must precede its associated data word. (In an array, the subscript of the point address variable must be one larger than the subscript of the data word variable.)

If only one data table is specified, the J parameter set must allow one extra variable position in the data table for the word count.

If chaining is specified, the first and last tables must allow two extra variable positions in the J parameter set. The two extra variables in the first table will be used for the word count and the chaining address, and in the last table for CAR and the word count. All tables between must allow three extra variable positions to be used for CAR, word count, and chaining address.

Example 1: (Figure 21)

```
DIMENSION     IDATA (5)
IDATA (4) = 84
IDATA (2) = 81
CALL CO (01101, IDATA (1), IDATA (5))
```

In this example, the I parameter (01101) specifies

0 - Random addressing
1 - Write operation
1 - Data channel control
0 - Not used, must be zero
1 - One data table and addresses

The J parameter (IDATA (1), IDATA (5)) specifies:

IDATA (1)	The last data word output to its associated address
IDATA (5)	The word count (4) of the table IDATA.

IDATA (5)	Word count (4) of IDATA
IDATA (4)	Address of point 84
IDATA (3)	Output data to point 84

● Figure 21. Random Address and Data Table

Example 2: (Figure 22)

```
DIMENSION     IDATA (5)
IDATA (4) = 84
CALL DO (11101, IDATA (1), IDATA (5))
```

In this example, the I parameter (11101) specifies:

1 - Single addressing
1 - Write operation
1 - Data channel control
0 - Not used, must be zero
1 - One data table and address

The J parameter (IDATA (1), IDATA (5)) specifies:

IDATA (1)	The last output data word to the addressed point.
IDATA (5)	Word count (4) of the table IDATA.

IDATA (5)	Word count (4) of IDATA
IDATA (4)	Address of point 84
IDATA (3)	First output data word to point 84
IDATA (2)	Second output data word to point 84
IDATA (1)	Last output data word to point 84

● Figure 22. Single Address and Data Table

Example 3: (Figure 23)

```
DIMENSION   IN1 (6), IN2 (7), IN3 (6)
IN1 (5) = 84
IN1 (3) = 81
IN2 (5) = 86
IN2 (3) = 64
IN3 (4) = 77
IN3 (2) = 74
CALL CO (01103, IN1 (1), IN1 (6), IN2 (1),
IN2 (7), IN3 (1), IN3 (6))
```

In this example, the I parameter (01103) specifies:

0 - Random Addressing
1 - Write operation
1 - Data channel control
0 - Not used, must be zero
3 - Three data tables and addresses

The J parameter (IN1 (1), IN1 (6), IN2 (1), IN2 (7), IN3 (1), IN3 (6)) specifies:

IN1 (1) The chaining address (IN2 (7)) that is stored in CAR of the second data table.

IN1 (6) Word count (4) of the table IN1.

IN2 (1) The chaining address (IN3 (6)) that is stored in CAR of the third data table.

IN2 (7) CAR where the chaining address (IN2 (7)) is stored.

IN3 (1) The last output data word of the chain of tables, to its associated address.

IN3 (6) CAR where the chaining address (IN3 (6)) is stored.

NOTE: The chaining methods described in Example 3 would be the same for single addressing. The only difference is the appearance of the data table, only three variable positions would be designated as address points.

IN1 (6)	Word count (4) of IN1
IN1 (5)	Address of point 84
IN1 (4)	Output data word to point 84
IN1 (3)	Address of point 81
IN1 (2)	Output data word to point 81
IN1 (1)	Chaining address (IN2 (7))

IN2 (7)	CAR (IN2 (7))
IN2 (6)	Word count (4) of IN2
IN2 (5)	Address of point 86
IN2 (4)	Output data word to point 86
IN2 (3)	Address of point 64
IN2 (2)	Output data word to point 64
IN2 (1)	Chaining address (IN 3 (6))

IN3 (6)	CAR (IN3 (6))
IN3 (5)	Word count (4) of IN3
IN3 (4)	Address of point 77
IN3 (3)	Output data word to point 77
IN3 (2)	Address of point 74
IN3 (1)	Output data word to point 74

Figure 23. Chained Address and Data Tables

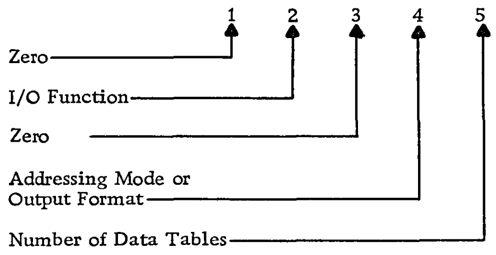
DIGITAL INPUT

The FORTRAN CALL statements shown below are used to call the digital input subroutines to execute contact sense, voltage level sense, digital input, and pulse counter operations.

CALL CS (I,J)	Contact Sense
CALL CSC (I,J,M)	Contact Sense and Compare
CALL CSX (I,J)	Contact Sense and Expand
CALL VS (I,J)	Voltage Level Sense
CALL VSC (I,J,M)	Voltage Level Sense and Compare
CALL VSX (I,J)	Voltage Level Sense and Expand
CALL DI (I,J)	Digital Input
CALL DIC (I,J,M)	Digital Input and Compare
CALL DIX (I,J)	Digital Input and Expand
CALL PI (I,J)	Pulse Counter Input
CALL PIC (I,J,M)	Pulse Counter Input and Compare
CALL PIX (I,J)	Pulse Counter Input and Expand

I Parameter

The I parameter is an integer constant or integer variable that specifies various control options. It consists of a 0 for the test function or 5 decimal digits for a read or read and compare function.



I/O Function

The I/O function digit specifies the function to be performed by the DINP, DICMP, DIEXP subroutines. The functions, associated digital values, and required parameters are listed and described below.

Function	Digital Value	Required Parameters	FORTRAN Subroutine Used	Subroutine Used
Test	0	I, J*	Any	DINP, DICPM, DIEXP
Read Data Channel	1	I, J	CS, VS, DI, PI	DINP
or Read and Compare	1	I, J, M	CSC, VSC, DIC, PIC	DICMP
or Read and Expand	1	I, J	CSX, VSX, DIX, PIX	DIEXP
Read Program Control	2	I, J	CS, VS, DI, PI	DINP

*The J parameter for the test function is a single integer variable (same as described under AIP).

Test. The test function determines if the requested device is busy. The J parameter is the same as described for the analog input single read subprogram (see AIP).

Read Data Channel. Reads the requested number of input values using data channel control. Used with CS, VS, DI, and PI only.

Read and Compare. Reads, under direct program control using sequential addressing, and compares input groups (words). Used with CSC, VSC, DIC, and PIC only.

Read and Expand. Reads the requested group of digital input points (i.e., one word at a time) and expands them into 2, 4, 8, or 16 words containing 8, 4, 2, or 1 bit each, respectively. Read and expand is used with CSX, VSX, DIX, and PIX only. The addressing mode is replaced by an output format digit for this function.

Read Program Control. Reads the requested number of input values, using program control and sequential addressing. Used with CS, VS, DI and PI only. The addressing mode/output format digit is not used.

Addressing Mode or Output Format

The addressing mode digit specifies one of six available addressing options for data channel operation:

0 - Random addressing
1 - Sequential addressing
2 - Single addressing
3 - Random address with external synchronization
4 - Sequential addressing with external synchronization
5 - Single addressing with external synchronization

The output format must be specified for the read and expand function only; 1, 2, 4, or 8 indicates 16 one-bit words, 8 two-bit words, 4 four-bit words, or 2 eight-bit words, respectively.

Number of Data Tables

This digit specified the number of pairs of variables that identify I/O area boundaries. Up to nine pairs of variables (nine I/O areas) can be specified with CS, VS, DI, or PI. This digit must correspond to the actual number of pairs specified by the J parameter. Only one pair can be used with CSC, CSX, VSC, VSX, DIC, DIX, PIC, AND PIX, or if the read program control function is specified, with CS, VS, DI, and PI.

J Parameter

The J parameter consists of one or more pairs of integer variables that define the data table area. There must be one pair of J parameters for each data table. If random addressing is used, each point address must precede its associated data word. If single or sequential addressing is used, there is only one digital point address and that address precedes all data words. (In an array, the subscript of the point address variable must be one larger than the subscript of the data word variable.) The first and last variables of a J parameter pair must specify the last and first variables of the data table, respectively.

If only one data table is specified, the J parameter must allow one extra variable position in the data table for the word count except if the function is read and expand. In that case, the number of words of expanded input is equal to the number of words of expanded data specified by the output format code, i.e., 16, 8, 4, or 2 words of expanded data (CSX, VSX, DIX, and PIX). Also, the first word of the table must be the digital input address.

If chaining is specified (more than one data table with data channel control), the first and last data tables must have two extra variable positions each. In the first table, the extra variables will be used for the word count and the chaining address. In the last table, the extra variables will be used for CAR and the word count. All tables between must have three extra variable positions that will be used for CAR, word count, and chaining address.

M Parameter

The M parameter specifies the name of the user-written subprogram to which control will be transferred upon comparison of unequal values. The subprogram name must also be listed in the

EXTERNAL statement. This subprogram must have two parameters in the calling sequence: one must be an integer variable that will be loaded with the current reading that did not provide an equal comparison; the other must be an integer variable that will be loaded with the address of the point group that did not provide an equal comparison. The M parameter is required only with CSC, VSC, DIC, and PIC calls.

Example 1: (Figure 24)

```
EXTERNAL     CLIM
DIMENSION    IN(12)


IN(11) = 87
CALL CSC (01001, IN(1), IN(12), CLIM)
```

In this example, the I parameter (01001) specifies:

0 - Not used, must be zero.
1 - Read and compare function.
0 - Must be zero.
0 - Not used.
1 - One data table.

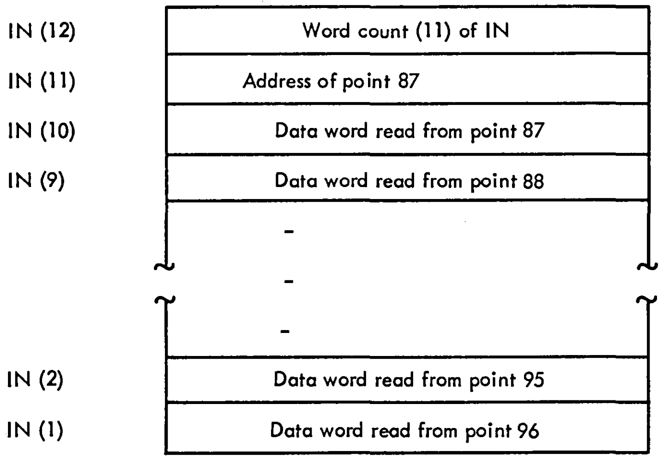
The J parameter (IN(1), IN(12)) specifies:

IN(1) The last data word read into the table from the point that is ten positions (sequentially) from the first address, point 87 in this case.
IN(12) The word count (11) of the data table IN.

The M parameter (CLIM) specifies:

CLIM This is the user-written subprogram to which the CSC call would transfer control upon a comparison of unequal values. The subprogram might have the two required parameters designated as LV and MX, where:
LV is the integer variable that will be loaded with the current reading that did not provide an equal comparison, and
MX is the integer variable that will be loaded with the address of the point group that did not provide an equal comparison.

NOTE: Except for the difference between a write operation and a read operation, the examples of multiple tables and chaining for digital/analog output apply to digital input.



IN (12)	Word count (11) of IN
IN (11)	Address of point 87
IN (10)	Data word read from point 87
IN (9)	Data word read from point 88
	-
	-
	-
IN (2)	Data word read from point 95
IN (1)	Data word read from point 96

Figure 24. Single Address and Data Table

Example 2: (Figure 24.1)

```
DIMENSION         IN(5)
IN(5) = 64
CALL CSX (01041, IN(1), IN(5))
```

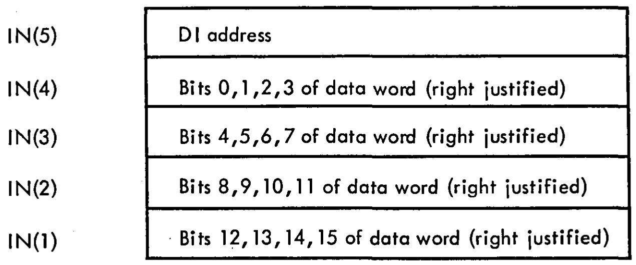
In this example, the I parameter (01041) specifies:

0 - Not used, must be zero.
1 - Read and compare function.
0 - Must be zero.
4 - Sequential point addressing.
1 - One data table.

The J parameter (IN(1), IN(5)) specifies:

IN(1) The last data word filled in.
IN(5) Digital input address.

Figure 24.1. Single Address and Data Table (CSX)



IN(5)	DI address
IN(4)	Bits 0,1,2,3 of data word (right justified)
IN(3)	Bits 4,5,6,7 of data word (right justified)
IN(2)	Bits 8,9,10,11 of data word (right justified)
IN(1)	Bits 12,13,14,15 of data word (right justified)

CONVERSION SUBROUTINES

The basic unit of information within the IBM 1800 Data Acquisition and Control System is the 16-bit binary word. This information may be interpreted in a variety of ways, depending on the circumstances. For example, in purely internal computer operations, computer words may be interpreted as instructions, addresses, binary integers, or floating-point numbers.

This section is concerned with the interpretations of bit configurations that relate computer information with the outside world. These interpretations are made necessary by the following considerations:

1. A compact notation to represent externally the bit configuration within each computer word is needed by the programmer. This is provided by hexadecimal notation.
2. A code is required to represent alphameric (mixed alphabetic and numeric) data within the computer. This is provided by the extended binary coded decimal interchange code (EBCDIC).
3. The design and operation of the various input/output devices is such that many of them impose a unique correspondence between character representations in the external medium and the associated bit configurations within the computer. Conversion subroutines are needed to convert inputs from these devices into a form on which the computer can operate, and to prepare computed results for output on the devices.

This section of the manual describes the subroutines for converting data representations among these various codes.

DATA CODES

In addition to the 16-bit binary internal representation, the conversion subroutines handle the following six codes:

1. Hexadecimal notation
2. IBM card code
3. 1443 Printer code
4. Perforated tape and transmission code (PTTC/8)
5. 1053 Printer code
6. Extended binary coded decimal interchange code (EBCDIC)

A list of these codes can be found in Appendix D.

Hexadecimal Notation

Although binary numbers facilitate the operations of computers, they are bulky and awkward for the programmer to handle. A long string of 1's and 0's cannot be effectively transmitted from one individual to another. The hexadecimal number system is often used as a shorthand method of communicating binary numbers. Because of the simple relationship of hexadecimal to binary, numbers can easily be converted from one system to another.

In hexadecimal notation a single digit is used to represent a four-bit binary value. The correspondence between binary, decimal, and hexadecimal is shown in Figure 25. Thus, a 16-bit word in the IBM 1800 system can be expressed as four hexadecimal digits. For example, the binary value

1101001110111011

can be separated into four sections.

BINARY	DECIMAL	HEXADECIMAL
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Figure 25. Hexadecimal Notation

Binary	1101/0011/1011/1011
Hexadecimal	D 3 B B

Another advantage of hexadecimal notation is that fewer positions are required when output data is printed, or punched in cards or paper tape. In the example above, only four card columns are required to contain the data from a 16-bit binary word.

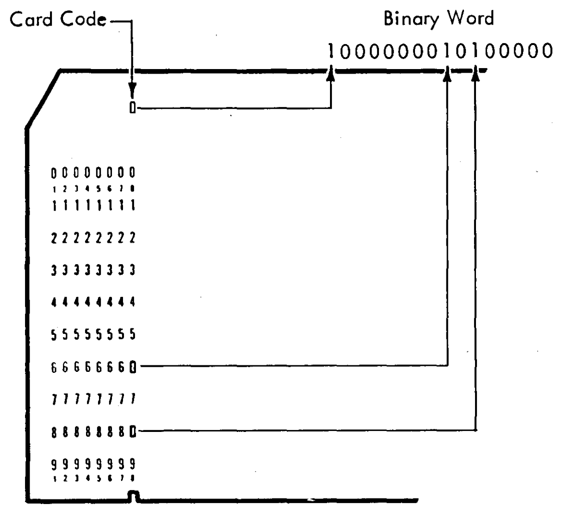
IBM Card Code

The IBM card code can be used as an input/output code with the 1442 Card Read/Punch, and is the input code for the 1816 Printer-Keyboard.

This code defines a character by a combination of punches in a card column. Card-code data is taken from, or placed into, the leftmost twelve bits of a computer word as shown below:

Card Row	12	11	0	1	2	3	4	5	6	7	8	9	-	-	-	-
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For example, a plus sign which has a card code of 12, 6, 8 is placed into core storage in the binary configuration illustrated in the following diagram.



1443 Printer Code (6-bit BCD)

In this code, all characters are represented by six positions of binary notation. These positions consist of two zone positions and four numeric positions as shown below.

Zone	Numeric
BA	8 4 2 1

The four numeric positions are assigned decimal values of 8, 4, 2, and 1. Combinations of zone and numeric bits represent alphabetic and special characters.

Each computer word must contain two printer characters as shown below:

Printer BCD Characters	-	-	B	A	8	4	2	1	-	-	B	A	8	4	2	1
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(Bits 0, 1, 8, and 9 are not used)

For example, the printer characters ? R are shown in Figure 26 as they would be represented by a 6-bit BCD code and as they are packed into a computer word.

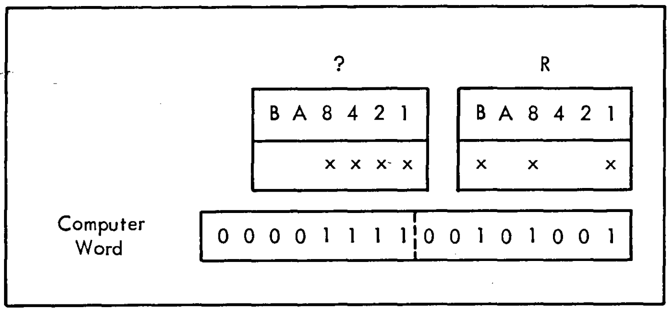
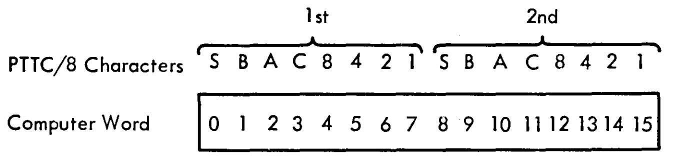


Figure 26. 1443 Printer Code for ?R

Perforated Tape and Transmission Code (PTTC/8)

The PTTC/8 code is an 8-bit code that can be used with the 1054/1055 Paper Tape units. This code represents a character with a stop position, a check position, and six positions representing the 6-bit code

BA8421. PTTC/8 characters can be packed two per computer word, as shown below.



The binary configuration of paper tape code for the characters ? R is shown in Figure 27.

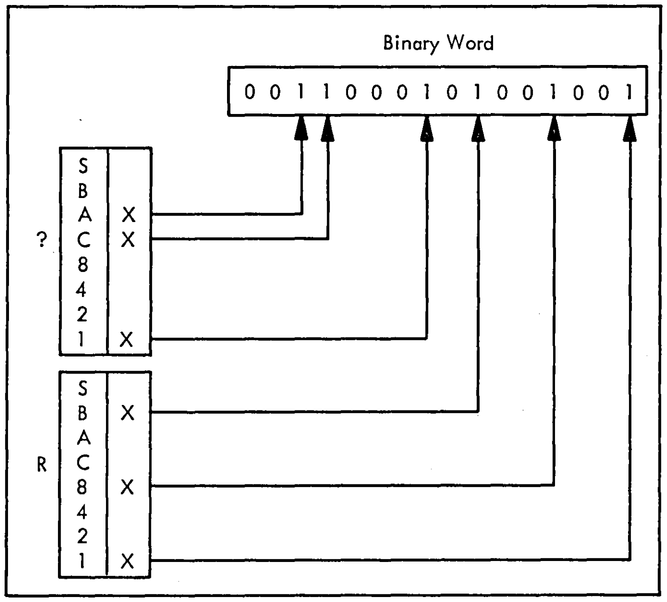


Figure 27. PTTC/8 Code for ?R

NOTE: The DEL and NL characters have a special meaning when encountered by the paper tape subroutine (check mode only).

1053 Printer code

This code is the 8-bit output code for the 1816 Printer-Keyboard and the 1053 Printer. Characters can be packed two per computer word.

NOTE: The following control characters have special meanings when used with the 1816 Printer-Keyboard and the 1053 Printer:

Character	Control Operation
HT	Tabulate
RES	Shift to black ribbon
NL	Carrier return on new line
BS	Backspace
LF	Line feed without carrier return
RS	Shift to red ribbon

Extended Binary Coded Decimal Interchange Code (EBCDIC)

EBCDIC is the standard code for internal representation of alphameric and special characters. The code occupies eight binary bits per character, making it possible to store one, or two characters per computer word. The eight bits allow 256 possible codes. (At present, not all of these combinations have been assigned to represent characters.) The complete EBCDIC code is shown in Appendix D.

Most of the conversion subroutines do not recognize all 256 codes. The asterisked codes in Appendix D constitute the subset which is recognized by most of the conversion subroutines.

DESCRIPTIONS OF CONVERSION SUBROUTINES

The following data conversion subroutines are provided:

BINDC Binary value to IBM card-coded decimal value.

DCBIN IBM card-coded decimal value to binary value.

BINHX Binary value to IBM card-coded hexadecimal value.

HXBIN IBM card-coded hexadecimal value to binary value.

HOLEB IBM card code subset to EBCDIC subset; EBCDIC subset to IBM card code subset.

PAPEB PTTC/8 subset to EBCDIC subset; EBCDIC subset to PTTC/8 subset.

PAPHL PTTC/8 subset to IBM card code subset, IBM card code subset to PTTC/8 subset.

PAPPR PTTC/8 subset to either 1443 or 1053 Printer code.

HOLPR IBM card code subset to either 1443 or 1053 Printer code.

EBPRT EBCDIC subset to either 1443 or 1053 Printer code.

In addition to the subroutines listed above, the following conversion tables are used by some conversion subroutines.

PRTY	Typewriter output and 1443 Printer codes.
EBPA	EBCDIC and PTTC/8 subsets.
HOLL	IBM Card code subset.

The first four of the conversion subroutines change numeric data from its input form to a binary form, or from a binary form to an appropriate output data code. The last seven convert entire messages, one character at a time, from one input/output code to another. The different types of conversions offered by these subroutines are illustrated in Table 6.

Error Checking

All of the subroutines will accept only the codes asterisked in Appendix D. It is considered an error if any input character does not belong to the specified input code. A space character in the output code, is stored in the output area in place of the input character in error.

If any such error occurs, bit one of word 55 of the work level is turned on when the conversion subroutine returns control to the user. Otherwise, the settings of the carry and overflow indicators are not altered by the subroutine.

It is the user's responsibility to turn off this bit before calling a conversion subroutine if he plans to test it later. This word can be reached by picking up the address in location 104 and adding 55 to it. (The contents of 104 must not be changed.)

BINDC

This subroutine converts a 16-bit binary value to its decimal equivalent in five IBM card-coded numerical characters and one sign character. The five characters and the sign are placed into six computer words as illustrated in Figure 28.

	CONVERTED TO							
CONVERTED FROM	Binary	IBM Card Code (Subset)	PTTC/8 (Subset)	EBCDIC (Subset)	1443 Printer	1053 Printer	Hex Equivalent (Card Code)	Decimal Equivalent (Card Code)
Binary							BINHX	BINDC
IBM Card Code (Subset)			PAPHL	HOLEB	HOLPR	HOLPR		
PTTC/8 (Subset)		PAPHL		PAPEB	PAPPR	PAPPR		
ESCDIC (Subset)		HOLEB	PAPEB		EBPRT	EBPRT		
HEX Equivalent (Card Code)	HXBIN							
Decimal Equivalent (Card Code)	DCBIN							

Table 6. Types of Conversions

Calling Sequence

```
            LIBF    BINDC
            DC      OUTPT
             .        .
             .        .
             .        .
OUTPT       BSS     6
```

Input. Input is a 16-bit binary value in the A-register.

Output. Output is an IBM card-coded sign character (plus or minus) in location OUTPT, and five IBM card-coded numerical characters in OUTPT +1 through OUTPT +5.

I/O Locations	Conversion Data	Bits in Core Storage 0 ◄——► 15
A - Register	+01538	0000 0110 0000 0010
OUTPT	+	1000 0000 1010 0000
OUTPT+1	0	0010 0000 0000 0000
OUTPT+2	1	0001 0000 0000 0000
OUTPT+3	5	0000 0001 0000 0000
OUTPT+4	3	0000 0100 0000 0000
OUTPT+5	8	0000 0000 0010 0000

Figure 28. BINDC Conversion

DCBIN

This subroutine converts a decimal value in five IBM card-coded characters and a sign character to a 16-bit binary word. The conversion is the reverse of the BINDC subroutine conversion illustrated in Figure 28.

Calling Sequence

```
            LIBF    DCBIN
            DC      INPUT
             .        .
             .        .
             .        .
             .        .
INPUT       BSS     6
```

Input. Input is an IBM card-coded sign character in location INPUT and five 12-bit IBM card-coded decimal characters in INPUT +1 through INPUT +5.

Output. Output is a 16-bit binary word in A-register, containing the converted value.

Error Conditions Detected

Any character other than an IBM card-coded plus, ampersand, space, or minus as the sign, and space or 0 through 9 as a decimal digit is considered an error. Any converted value greater than +32767 or less than -32768 is considered an error.

BINHX

This subroutine converts a 16-bit binary word into hexadecimal notation in four IBM card-coded characters as illustrated in Figure 29.

Calling Sequence

```
            LIBF    BINHX
            DC      OUTPT
             .        .
             .        .
             .        .
             .        .
OUTPT       BSS     4
```

Input. Input is a 16-bit binary word in the A-register.

I/O Locations	Conversion Data	Bits in Core Storage 0 ◄——► 15
A - Register	A59E	1010 0101 1001 1110
OUTPT	A	1001 0000 0000 0000
OUTPT+1	5	0000 0001 0000 0000
OUTPT+2	9	0000 0000 0001 0000
OUTPT+3	E	1000 0001 0000 0000

Figure 29. BINHX Conversion

Output. Output is four IBM card-coded hexadecimal digits in location OUTPT through OUTPT +3.

HXBIN

This subroutine converts four IBM card-coded hexadecimal characters into one 16-bit binary word. The conversion is the reverse of the BINHX subroutine conversion illustrated in Figure 29.

Calling Sequence

```
                LIBF      HXBIN
                DC        INPUT
                 .          .
                 .          .
                 .          .
                 .          .
                 .          .
      INPUT     BSS         4
```

Input. Input is four IBM card-coded hexadecimal digits in INPUT through INPUT +3.

Output. Output is a 16-bit binary word in the A-register.

Error Conditions Detected

Any character other than IBM card-coded 0 through 9 or A through F is considered an error.

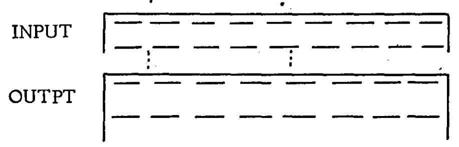
HOLEB

This subroutine converts IBM card code subset to EBCDIC subset or converts EBCDIC subset to IBM card code subset. This code conversion is illustrated in Figure 30.

Only the character codes asterisked in Appendix D can be converted by this subroutine.

Calling Sequence

```
              LIBF      HOLEB
              DC        /xxxx     (Control)
              DC        INPUT
              DC        OUTPT
              DC        nnnnn     (Character count)
               :          :
```



I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	JS	1101 0001 1110 0010
OUTPT	J	0101 0000 0000 0000
OUTPT +1	S	0010 1000 0000 0000

Figure 30. HOLEB Conversion (EBCDIC to Card Code)

Control Parameter. This parameter consists of four hexadecimal digits. Digits 1-3 are not used: the fourth digit specifies the direction of conversion:

0 - IBM card code to EBCDIC
1 - EBCDIC to IBM card code.

Input. Input is either IBM card code or EBCDIC characters (as specified by the control parameter) starting in location INPUT. EBCDIC characters must be packed two characters per one binary word. IBM card-coded characters are stored one character to each binary word.

Output. Output is either IBM card code or EBCDIC characters starting in location OUTPT. Characters are packed as described above.

If the direction of the conversion is IBM card code input to EBCDIC output, the input area can overlap the output area if the address INPUT is equal to or greater than the address OUTPT. If the direction of the conversion is EBCDIC input to IBM card code output, the input area can overlap the output area if the address INPUT + (n/2) -1 is equal to or greater than the address OUTPT +n-1, where n is the character count specified. The subroutine starts processing at location INPUT.

Character Count. This parameter specifies the number of characters to be converted; it is not equal to the number of binary words used for the EBCDIC characters because those characters are packed two per binary word. If an odd count is specified, and the output code is EBCDIC, bits 8 through 15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which is not asterisked in Appendix D is considered an error.

PAPEB

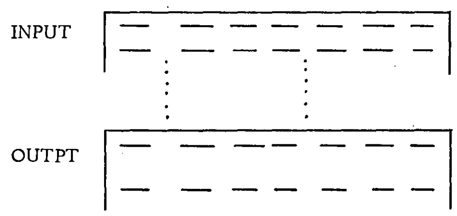
This subroutine converts PTTC/8 subset to EBCDIC subset or converts EBCDIC subset to PTTC/8 subset. The conversion is illustrated in Figure 31.

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	JS	1101 0001 1110 0010
OUTPT +0	UC J	0000 1110 0101 0001
+1	S DEL	0011 0010 0111 1111

Figure 31. PAPEB Conversion (EBCDIC to PTTC/B)

Calling Sequence

```
LIBF    PAPEB
DC      /xxxx    (Control
DC      INPUT
DC      OUTPT
DC      nnnnn    (Character count)
  .       .
  .       .
```



Control Parameter. This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether the case is to be initialized for the first graphic character before conversion begins.

0 - Initialize case
1 - Do not alter

The fourth digit indicates the direction of conversion.

0 - PTTC/8 to EBCDIC
1 - EBCDIC to PTTC/8

Input. Input is either PTTC/8 or EBCDIC characters, as specified by the control parameter, starting in location INPUT. Both character codes are packed two per computer word.

If the input is in PTTC/8 code, all control characters except case shift (LC or UC) characters are converted to output. Case shift characters are used to define the case mode of the graphic characters which follow.

If the input is in EBCDIC, all data and control characters are converted to output. The user should not specify case shifting in his input message; this is handled automatically by the PAPEB subroutine. If a case shift control character appears in the input message, it may appear twice in the output.

Output. Output is either EBCDIC or PTTC/8 characters, starting in OUTPT. Both character codes are in packed format. The subroutine starts processing at location INPUT.

If the output is in EBCDIC, overlap of the input and output areas is possible if the address INPUT is equal to or greater than the address OUTPT.

If the output is in PTTC/8, overlap of the input and output areas is not recommended because the number of output characters might be greater than the number of input characters.

Case shift characters are inserted in a PTTC/8 output message where needed to define certain graphic characters. This is necessary because some graphic characters have the same binary value and are differentiated only by a case mode character. For example, the binary value 0101 1011 (5B), will be interpreted as an $ in lower case and an ! in upper case (see Appendix D).

Character Count. This parameter specifies the number of PTTC/8 or EBCDIC characters in the input area. The count must include case shift characters even though they will not appear in the output. Because the input is packed, the character count will not be equal to the number of binary words in the input area.

If an odd number of output characters is produced, bits 8-15 of the last word used in the output area are set to a space character if the output is EBCDIC or to a delete character if the output is PTTC/8.

The conversion is halted whenever the character count is decremented to zero or a new line (NL) control character is detected.

Error Conditions Detected

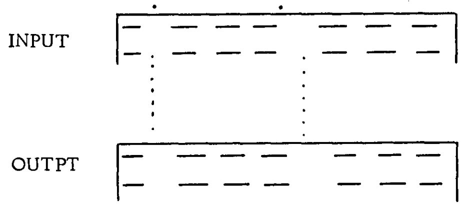
Any input character which is not asterisked in Appendix D is considered an error.

PAPHL

This subroutine converts PTTC/8 subset to IBM card code subset or IBM card code subset to PTTC/8 subset. Figure 32 illustrates the relationship of the two codes for converting PTTC/8 to IBM card code.

Calling Sequence

```
        LIBF     PAPHL
        DC       /xxxx    (Control)
        DC       INPUT
        DC       OUTPT
        DC       nnnnn    (Character count)
         :         :
```



Control Parameter. This parameter consists of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether or not the case is to be initialized for the first graphic character before conversion begins.

0 - Initialize case
1 - Do not alter

I/O Locations	Conversion Data	Bits in Core Storage 0 → 15
INPUT	UC J	0000 1110 0101 0001
	S T	0011 0010 0010 0011
OUTPT	J	0101 0000 0000 0000
OUTPT +1	S	0010 1000 0000 0000
OUTPT +2	T	0010 0100 0000 0000

Figure 32. PAPHL Conversion (PTTC/B to Card Code)

The fourth digit indicates the type of conversion.

0 - PTTC/8 to IBM card code
1 - IBM card code to PTTC/8

Input. Input is either PTTC/8 or IBM card code characters, as specified by the control parameter, starting in location INPUT. PTTC/8 characters are packed two per binary word; IBM card code characters are not packed.

If the input is in PTTC/8 code, all control characters, except case shift (LC or UC) characters are converted to output. Case shift characters are used simply to define the case mode of the graphic characters which follow.

If the input is in IBM card code, all data and control characters are converted to output. The user should not specify case shifting in the input message. This is handled automatically by the PAPHL subroutine. If a case shift control character appears in the input message, it may appear twice in the output.

Output. Output is either IBM card code or PTTC/8 code characters, starting in location OUTPT. PTTC/8 codes are packed two per binary word; IBM card code characters are not packed.

If the direction of the conversion is IBM card code input to PTTC/8 output, the input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. Case shift characters are inserted in the output message where needed to define certain graphic characters (see PAPEB).

If the direction of the conversion is PTTC/8 input to IBM card code output, the input area may overlap the output area if the address INPUT + (n/2) is equal to or greater than the address OUTPT + n, where n is the character count. The subroutine starts processing at location INPUT.

Character Count. This parameter specifies the number of PTTC/8 or EBCDIC characters in the input area. The count must include case shift characters even though they might not appear in the output. Because the input can be packed, the character count may not be equal to the number of binary words in the input area.

If an odd number of PTTC/8 output characters is produced, bits 8-15 of the last word used in the output area are set to a delete character.

The conversion is halted whenever the character count is decremented to zero or a new line (NL) control character is detected.

Error Conditions Detected

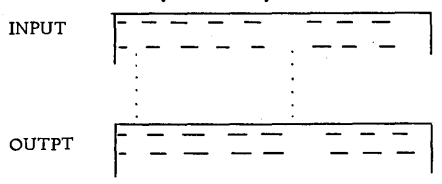
Any input character which is not asterisked in Appendix D is considered an error.

PAPPR

This subroutine converts PTTC/8 subset to either 1053 or 1443 Printer code. The conversion to 1443 Printer code is illustrated in Figure 33.

Calling Sequence

```
        LIBF    PAPPR
        DC      /xxxx    (Control)
        DC      INPUT
        DC      OUTPT
        DC      nnnnn    (Character count)
         .        .
         .        .
```



Control Parameter. This parameter consits of four hexadecimal digits. Digits 1 and 2 are not used. The third digit indicates whether or not the case is to be initialized for the first graphic character before conversion begins.

0 - Initialize case
1 - Do not alter

The fourth digit determines the output code.

0 - 1053 Printer code
1 - 1443 Printer code

I/O Locations	Conversion Data	Bits in Core Storage 0⟵⟶15
INPUT	UC J	0000 1110 ¦ 0101 0001
INPUT +1	LC $	0110 1110 ¦ 0101 1011
OUTPT	J $	0111 1100 ¦ 0100 0000

Figure 33. PAPPR Conversion (PTTC/B to 1443 Printer Code)

Input. Input is PTTC/8 characters, starting in location INPUT. PTTC/8 characters are packed two per binary word.

All control characters, except case shift (LC or UC) characters, are converted to output. Case shift characters are used to define the case mode of the graphic characters which follow.

Output. Output is either 1443 or 1053 Printer characters, starting in location OUTPT. Both codes are packed two per binary word.

If overlap of the input and output areas is desired, the address INPUT must be equal to or greater than the address OUTPT. This is necessary because the subroutine starts processing at location INPUT.

Character Count. This parameter specifies the number of PTTC/8 characters in the input area. The count must include case shift characters even though they will not appear in the output. Because the input is packed, the character count will not be equal to the number of binary words in the input area.

If an odd number of output characters is produced, bits 8-15 of the last word used in the output area are set to a space character.

The conversion is halted whenever the character count is decremented to zero or a new line (NL) control character is detected.

Error Conditions Detected

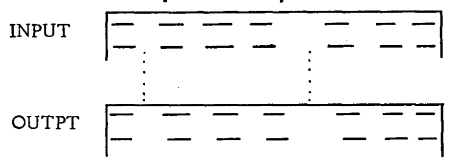
Any input character which is not asterisked in Appendix D is considered an error.

HOLPR

This subroutine converts IBM card code subset to either 1443 or 1053 Printer characters. The conversion to the 1053 Printer characters is illustrated in Figure 34.

Calling Sequence

```
        LIBF    HOLPR
        DC      /xxxx    (Control
        DC      INPUT
        DC      OUTPT
        DC      nnnnn    (Character count)
         :        :
```



Control Parameter. This parameter consists of four hexadecimal digits. Digits 1-3 are not used; the fourth digit determines the output code.

0 - 1053 Printer code
1 - 1443 Printer code

Input. Input is IBM card code characters starting in location INPUT. The characters are not packed.

Output. Output is either 1443 or 1053 Printer characters, starting in location OUTPT. Both codes are packed two per binary word.

The input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. The subroutine starts processing at location INPUT.

I/O Locations	Conversion Data	Bits in Core Storage 0 ◄——► 15
INPUT	J	0101 0000 0000 0000
INPUT+1	?	0010 0000 0110 0000
INPUT+2	W	0010 0000 1000 0000
OUTPT	J?	0111 1100 1000 0110
OUTPT+1	W	1001 0000 0000 0000

Figure 34. HOLPR Conversion (Card Code to 1053 Printer Code)

Character Count. This parameter specifies the number of IBM card code characters to be converted. This count is equal to the number of words in the input area. If an odd count is specified, bits 8-15 of the last word used in the output area are not altered.

Error Conditions Detected

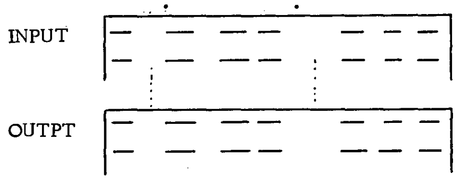
Any input character which is not asterisked in Appendix D is considered an error.

EBPRT

This subroutine converts EBCDIC subset to either 1443 or 1053 Printer characters. The conversion to 1443 Printer code is shown in Figure 35.

Calling Sequence

```
        LIBF    EBPRT
        DC      /xxxx    (Control)
        DC      INPUT
        DC      OUTPT
        DC      nnnnn    (Character count)
         :        :
```



Control Parameter. This parameter consists of four hexadecimal digits. Digits 1-3 are not used; the fourth digit determines the output code.

0 - 1053 Printer code
1 - 1443 Printer code

Input. Input is EBCDIC characters, starting in location INPUT. EBCDIC characters are packed two per word.

Output. Output is either 1443 or 1053 Printer characters, starting in location OUTPT. Both codes are packed two per binary word.

The address INPUT must be equal to or greater than the address OUTPT if overlap of the input and output areas is desired. The subroutine starts processing at location INPUT.

Character Count. This parameter specifies the number of EBCDIC characters to be converted. This count is not equal to the number of words in the input area. If an odd count is specified, bits 8-15 of the last used word in the output area are not altered.

Error Conditions Detected

Any input character which is not asterisked in Appendix D is considered an error.

I/O Locations	Conversion Data	Bits in Core Storage 0 ⟷ 15
INPUT	LE	1101 0011 1100 0101
INPUT+1	ES	1100 0101 1110 0010
OUTPT	LE	0010 0011 0011 0101
OUTPT+1	ES	0011 0101 0001 0010

Figure 35. EBPRT Conversion (EBCDIC to 1443 Printer Code)

ARITHMETIC AND FUNCTIONAL SUBROUTINES

The TSX subroutine library includes a selection of arithmetic and functional subroutines which are most frequently required because of their general applicability. There are 28 basic subroutines, some of which have several entry points. The various additional entry points allow indexed linkage, and/or a choice of format when working with floating-point numbers.

Table 7 lists the arithmetic and functional subroutines that are included in the subroutine library. After a brief description of data formats, the particulars of each subroutine are presented.

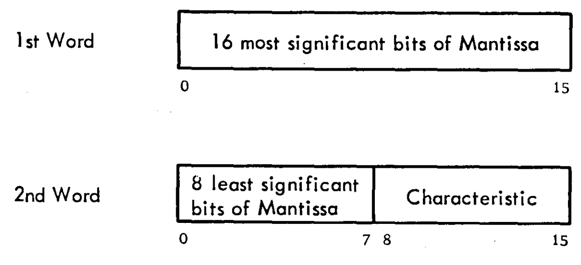
FLOATING-POINT DATA FORMATS

Many of the arithmetic and functional subroutines offer two ranges of precision. These ranges are called standard range and extended range. The standard range provides 23 bits of precision, while the extended range provides up to 31 bits of precision.

To achieve correct results from a particular subroutine, the input arguments must be in the proper format.

Standard Precision Format

Standard precision floating-point numbers are stored in core storage as shown below:

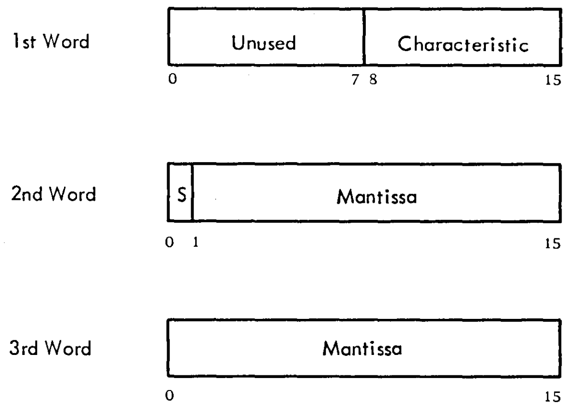

1st Word
16 most significant bits of Mantissa
0
15
2nd Word
8 least significant bits of Mantissa
Characteristic
0
7 8
15

Numbers can consist of up to 23 significant bits with a binary exponent ranging from -128 to +127. Two adjacent storage locations are required for each number. The first (lowest) location must be even-numbered. The sign of the mantissa is contained in bit zero of the first word. The next 23 bits represent the mantissa (2's complement) and the remaining 8 bits represent the characteristic. The mantissa is normalized to fractional form, i.e., the implied decimal point is between bits zero and one.

The characteristic is formed by adding +128 to the exponent. For example, an exponent of -32 would be represented by a characteristic of 128-32 or 96. An exponent of +100 would be represented by a characteristic of 100 + 128 or 228. Since $128_{10} = 200_8$ (80_{16}) the characteristic of a non-negative exponent always has a 1-bit in position 1, while the characteristic of a negative exponent always produces a 0-bit in position 1. A normal zero consists of all zero bits in both the characteristic and the mantissa.

Extended Precision Format

Extended precision floating-point numbers are stored in three adjacent core locations as shown in the following illustration.



Numbers can consist of up to 31 significant bits with a binary exponent ranging from -128 to +127. Bits zero through seven of the first word are unused; bits eight through 15 of the first word represent the characteristic of the exponent (formed in the same manner as in the standard precision format); bit zero of the second word contains the sign of the mantissa; and the remaining 31 bits represent the mantissa (2's complement).

Negative Number Representation

Negative numbers differ from positive numbers in one respect only. The mantissa is always the 2's

Table 7. Arithmetic and Functional Subroutines

SUBROUTINE	NAME	
Floating-Point	Standard Precision	Extended Precision
Add/Subtract	*FADD/*FSUB	*EADD/*ESUB
Multiply	*FMPY	*EMPY
Divide	*FDIV	*EDIV
Load/Store FAC	*FLD/*FSTO	*ELD/*ESTO
Trigonometric Sine/Cosine	FSINE/FCOSN, FSIN/FCOS	ESINE/ECOSN, ESIN/ECOS
Trigonometric Arctangent	FATN, FATAN	EATN, EATAN
Square Root	FSQR, FSQRT	ESQR, ESQRT
Natural Logarithm	FLN, FALOG	ELN, EALOG
Exponential (e^x)	FXPN, FEXP	EXPN, EEXP
Hyperbolic Tangent	FTNH, FTANH	ETNH, ETAHN
Floating-Point Base to an Integer Exponent	*FAXI	*EAXI
Floating-Point Base to a Floating-Point Exponent	*FAXB	*EAXB
Floating-Point to Integer	IFIX	IFIX
Integer to Floating-Point	FLOAT	FLOAT
Normalize	NORM	NORM
Floating Binary to Decimal/Floating Decimal to Binary	FBTD/FDTB	FBTD/FDTB
Floating-Point Arithmetic Range Check	FARC	FARC
Integer Base to an Integer Exponent	FIXI	FIXI
Fixed-Point Square Root	XSQR	XSQR
Fixed-Point Fractional Multiply (short)	XMDS	
Fixed-Point Double-Word Multiply	XMD	XMD
Fixed-Point Double-Word Divide	XDD	XDD
Special Function		
Floating-Point Reverse Subtract	FSBR	*ESBR
Floating-Point Reverse Divide	FDVR	*EDVR
Floating-Point Reverse Sign	SNR	SNR
Floating-Point Absolute Value	FAVL, FABS	EAVL, EABS
Integer Absolute Value	IABS	IABS

NOTE: By adding an X to those names prefixed with an asterisk, the user can cause the contents of the index register 1 to be added to the address of the argument specified in the subroutine calling sequence to form the effective address. For example, FADDX would be the modified form of FADD.

complement of the equivalent positive value. For example: (1) +.53125 is represented in core storage as 44000080; -.53125 is represented as BC000080, (2) +4.0 is represented in core storage as 40000083; -4.0 is represented as C0000083. A negative number can never be represented by a mantissa value of 800000xx. This number is its own 2's complement and therefore lies outside the definition of a negative number as the 2's complement of the absolute value.

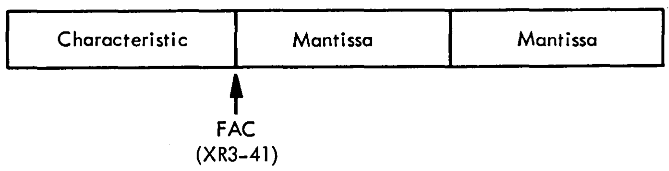
FIXED-POINT FORMAT

Fractional numbers, as applied to the fixed-point subroutines, XSQR, XMDS, XMD, and XDD, are defined as binary fractions with implied binary points of zero. That is, the binary point is positioned between the sign (bit 0) and the most significant bit (bit 1). The user can consider the binary point to be in any position in his fixed-point numbers. To correctly interpret the results, the following rules must be observed.

1. Only numbers with binary points in equivalent positions can be correctly added or subtracted.
2. The binary point location in the product of two numbers is the sum of the binary point locations of the multiplier and the multiplicand.
3. The binary point location in the quotient of two numbers is the difference between the binary point locations of the dividend and the divisor.
4. The binary point location in a number that is the input to the fixed-point square root subroutine (XSQR) must be an even number from 0-14. The binary point location in the root is half the binary point location of the input number.

FLOATING-POINT PSEUDO-ACCUMULATOR

The floating-point subroutines used in the 1800 TSX system sometimes require a register or accumulator that can accommodate numbers in floating-point format. These subroutines can be used at any level (i.e., mainline, interrupt, and nonprocess). Therefore, a register or accumulator must be provided for each level that can call a floating-point subroutine. These accumulators are located in the work levels of the system director. They are designated as the floating accumulator (FAC). The FAC is a three-word register that occupies the word 41, 42, and 43 of the work levels.



NOTE: The effective address of the mantissa will always be even.

PROGRAMMING CONSIDERATIONS

Subroutines which use the machine Registers A and Q do not save and restore its contents. Therefore, a main program should save the contents of the A-register, if the A-register is to be used.

CALLING SEQUENCES

The arithmetic and functional subroutines are called via a CALL or LIBF statement, in some cases followed by a DC statement containing the actual or symbolic address of an argument. In the descriptions which follow, the notations (ARG) and (FAC) refer to the contents of the operand rather than its address. The name FAC refers to the floating-point pseudo-accumulator. The extended precision subroutine names are prefixed with the letter E (subroutines which handle both precisions have the same name and do not have a prefix).

Note that some of the functional subroutines may be called via two different calling sequences. One calling sequence assumes the argument is in FAC, while the other specifies the location of the argument with a DC statement.

In addition, some subroutines can have indexed linkage to the argument. The calling sequence is the same except for the subroutine name which contains an X suffix. Also, some subroutines perform more than one type of arithmetic or function. For example, FSIN and FCOS are different entry points to the same subroutine. Each subroutine is listed in Table 4 with the corresponding entry points.

Floating-Point Add

LIBF	FADD, FADDX, EADD or EADDX
DC	ARG
Input	Floating-Point augend in FAC
	Floating-Point addend in location ARG
Result	(FAC) + (ARG) replaces (FAC)

Floating-Point Subtract

LIBF	FSUB, FSUBX, ESUB, ESUBX
DC	ARG
Input	Floating-Point minuend in FAC
	Floating-Point subtrahend in location ARG
Result	(FAC) - (ARG) replaces (FAC)

Floating-Point Multiply

LIBF	FMPY or EMPY
DC	ARG
Input	Floating-point multiplicand in FAC Floating-point multiplier in location ARG
Result	(FAC) times (ARG) replaces (FAC)

Floating-Point Divide

LIBF	FDIV, FDIVX, EDIV, or EDIVX
DC	ARG
Input	Floating-point dividend in FAC Floating-point divisor in location ARG
Result	(FAC)/(ARG) replaces (FAC)

Load FAC

LIBF	FLD, FLDX, ELD or ELDX
DC	ARG
Input	Floating-point number in location ARG
Result	(ARG) replaces (FAC)

Store FAC

LIBF	FSTO, ~~ESTO,~~ FSTOX, ESTO or ESTOX
DC	ARG
Input	Floating-point number in FAC
Result	(FAC) replaces (ARG)

Floating-Point Trigonometric Sine

CALL	FSINE or ESINE
Input	Floating-point argument (in radians) in FAC
Result	Sine of (FAC) replaces (FAC)

or

CALL	FSIN or ESIN
DC	ARG
Input	Floating-point argument (in radians) in location ARG
Result	Sine of (ARG) replaces (FAC)

Floating-Point Trigonometric Cosine

CALL	FCOSN or ECOSN
Input	Floating-point argument (in radians) in FAC
Result	Cosine of (FAC) replaces (FAC)

or

CALL	FCOS or ECOS
DC	ARG
Input	Floating-point argument (in radians) in location ARG
Result	Cosine of (ARG) replaces (FAC)

Floating-Point Trigonometric Arctangent

CALL	FATN or EATN
Input	Floating-point argument in FAC
Result	Arctangent of (FAC) replaces (FAC); the result lies within the range $\pm \pi/2$ radians

or

CALL	FATAN or EATAN
DC	ARG
Input	Floating-point argument in location ARG
Result	Arctangent of (ARG) replaces (FAC); the result lies within the range $\pm \pi/2$ radians

Floating-Point Square Root

CALL	FSQR or ESQR
Input	Floating-point argument in FAC
Result	Square root of (FAC) replaces (FAC)

or

CALL	FSQRT or ESQRT
DC	ARG
Input	Floating-point argument in location ARG
Result	Square root of (ARG) replaces (FAC)

Floating-Point Natural Logarithm

CALL	FLN or ELN
Input	Floating-point argument in FAC
Result	LOG_e (FAC) replaces (FAC)

or

CALL	FALOG or EALOG
DC	ARG
Input	Floating-point argument in location ARG
Result	Log_e (ARG) replaces (FAC)

Floating-Point Exponential

CALL	FXPN or EXPN
Input	Floating-point argument in FAC = n
Result	e^n replaces (FAC)
	or
CALL	FEXP or EEXP
DC	ARG
Input	Floating-point argument in location ARG = n
Result	e^n replaces (FAC)

Floating-Point Hyperbolic Tangent

CALL	FTNH or ETNH
Input	Floating-point argument in FAC
Result	TANH (FAC) replaces (FAC)
	or
CALL	FTANH or ETANH
DC	ARG
Input	Floating-point argument in location ARG
Result	TANH (ARG) replaces (FAC)

Floating-Point Base to an Integer Exponent

LIBF	FAXI, FAXIX, EAXI or EAXIX
DC	ARG
Input	Floating-point base in FAC Integer exponent in location ARG
Result	(FAC), raised to the exponent contained in ARG, replaces (FAC)

Floating-Point Base to a Floating-Point Exponent

CALL	FAXB, FAXBX, EAXB or EAXBX
DC	ARG
Input	Floating-point base in FAC Floating-point exponent in location ARG
Result	(FAC), raised to the exponent contained in ARG, replaces (FAC)

Floating-Point to Integer

LIBF	IFIX
Input	Floating-point number in FAC
Result	Integer in the A-register

Integer to Floating-Point

LIBF	FLOAT
Input	Integer in the A-register
Result	Floating-point number in FAC

Normalize

LIBF	NORM
Input	Floating point unnormalized number in FAC
Result	The mantissa portion of FAC is shifted until the most significant bit resides in bit position 1. The characteristic is changed to reflect the number of bit positions shifted.

Floating Binary to Decimal

CALL	FBTD
DC	LDEC
Input	Floating-point number in FAC
Result	A string of EBCDIC-coded data starting at location LDEC. Each EBCDIC character occupies the rightmost 8 bits of a word.

The output format is exactly as follows:

sd.dddddddEsdd

where s represents a sign (plus or minus) and d represents one of the decimal digits 0-9.

Floating Decimal to Binary

CALL	FDTB
DC	LDEC
Input	Same as output from FBTD subroutine. The input field may not contain any embedded blanks. The first blank encountered is interpreted as the end of the string.
Result	Floating-point number in FAC

Floating-Point Arithmetic Range Check

LIBF	FARC

Result	This subroutine checks for floating-point overflow or underflow, and sets programmed indicators for interrogation by a FORTRAN program.

Integer Base to an Integer Exponent

LIBF	FIXI or FIXIX
DC	ARG
Input	Fixed-point base in the A-register Fixed-point exponent in location ARG
Result	(A-register) raised to the exponent contained in ARG replaces (A-register)

Fixed-Point Square Root

CALL	XSQR
Input	Fixed-point fractional argument (16 bits only) in the A-register
Result	Square root of (A-register) replaces (A-register). If the argument is negative, the absolute value is used and the Overflow indicator is turned on.

Fixed-Point Fractional Multiply (short)

LIBF	XMDS
~~DC~~	~~ARG~~
Input	Double-word fractional multiplicand in the A- and Q-registers. Double-word fractional multiplier in FAC (addressed by XR3 + 41).
Result	Product in the A- and Q-register (XMDS is shorter and faster than XMD, however, the resulting precision is 24 bits).

Fixed-Point Double-Word Multiply

LIBF	XMD
Input	Double-word fractional multiplier in the FAC (addressed by XR3 + 41) Double-word fractional multiplicand in the A- and Q-registers.
Result	Double-word fractional product in the A- and Q-registers.

Fixed-Point Double-Word Divide

LIBF	XDD
Input	Double-word fractional dividend in the FAC (addressed by XR3 + 41) Double-word fractional divisor in the A- and Q-register
Result	Double-word fractional quotient in the A- and Q-registers. The double-word dividend in FAC is destroyed by the execution of the subroutine.

Floating-Point Reverse Subtract

LIBF	FSBR, FSBRX, ESBR or ESBRX
DC	ARG
Input	Floating-point minuend in location ARG Floating-point subtrahend in FAC
Result	(ARG) – (FAC) replaces (FAC)

Floating-Point Reverse Divide

LIBF	FDVR, FDVRX, EDVR or EDVRX
DC	ARG
Input	Floating-point dividend in location ARG Floating-point divisor in FAC
Result	(ARG)/(FAC) replaces (FAC)

Floating-Point Reverse Sign

LIBF	SNR
Input	Floating-point number, X, in FAC
Result	-X replaces X in FAC

Floating-Point Absolute Value

CALL	FAVL or EAVL
Input	Floating-point number, X, in FAC
Result	Absolute value of X replaces X in FAC

or

CALL	FABS or EABS
DC	ARG
Input	Floating-point number, X, in location ARG
Result	Absolute value of X replaces (FAC)

Integer Absolute Value

~~CALL~~	~~IABS~~
~~Input~~	~~An integer, I, in the A-register~~
~~Result~~	~~Absolute vlue of I replaces I in the A-register.~~

CALL	IABS
DC	ARG
Input	An integer in ARG
Result	Absolute value of (ARG) replaces (A-register)

ARITHMETIC AND FUNCTIONAL SUBROUTINE ERROR INDICATORS

Words 55, 56, 57 for each level are reserved for the arithmetic and functional subroutine error indicators.

Word 57 (addressed XR3 + 57) is used for floating-point arithmetic overflow and underflow indicators. Word 56 (XR3 + 56) is used for a divide check indicator, and the word 55 (XR3 + 55) is used for functional subroutine indicators. It is the user's responsibility to ensure that the indicators are reset before they are used.

Word 57

Each floating point subroutine checks for exponent underflow and overflow. If either occurs, word 57 and FAC are set as follows.

1, if overflow has occurred (FAC = ± maximum).
3, if underflow has occurred (FAC = zero).

The last error condition replaces any previous indication. Also, when an underflow occurs, FAC is set to zero.

When an overflow occurs, FAC is set to the largest valid number of the same algebraic sign as the contents of FAC when the overflow was detected.

Word 56

The floating-point divide subroutines check for division by zero. If this occurs, word 56 is set to 1. The dividend is not changed.

Word 55

The functional subroutines check for the following error conditions and set word 55 as described.

Floating-Point Square Root

When the argument is negative, the square root of the argument's absolute value is returned, and a bit is ORed into position 13 of word 55.

Floating-Point Natural Logarithm

When the argument is zero, FAC is set to the largest negative value and a bit is ORed into position 15 of word 55. When the argument is negative, the absolute value of the argument is used and a bit is ORed into position 15 of word 55.

Floating-Point Trigonometric Sine and Cosine

When the value of the sine argument lies outside the range $-1.0 \times 2^{23} + 1 <$ sin argument $< 1.0 \times 2^{23} - 1$ or the value of the cosine argument lies outside the range $-1.0 \times 2^{23} + 1 + \pi/2 <$ cos argument $< 1.0 \times 2^{23} - 1 + \pi/2$, FAC is set to zero and a bit is ORed into position 14 of word 55.

Floating-Point to Integer

When the absolute value of the argument is greater than $2^{15} - 1$, the largest possible signed result is placed in the A-register and a bit is ORed into position 12 of word 55.

Integer Base to an Integer Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 11 of word 55.

Floating-Point Base to an Integer Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 10 of word 55.

Floating-Point Base Raised to a Floating-Point Exponent

When the base is zero and the exponent is zero or negative, a zero result is returned and a bit is ORed into position 9 of word 57. When the base is negative and the exponent is not zero, the absolute value of the base is used and a bit is ORed into position 15 of word 55.

FUNCTIONAL SUBROUTINE ACCURACY

Given:

e	≡	Maximum error
f (x)	≡	True value of the function
f* (x)	≡	Value generated by subroutine
$(<+\infty)$	≡	≤ Largest valid floating-point number
$(>-\infty)$	≡	≥ Most negative floating-point number

EXTENDED PRECISION SUBROUTINES

The following statements of accuracy apply to extended precision subroutines.

ESIN

$$e \equiv \left| \frac{\sin(x) - \sin^*(x)}{x} \right| < 3.0 \times 10^{-9}$$

for the range

$$-1.0 \times 10^6 \leq x < 0$$

$$1.0 \times 10^6 \geq x > 0$$

for $x = 0$ $\sin(x) = 0$

ECOS

$$e \equiv \left| \frac{\cos(x) - \cos^*(x)}{|x| + \frac{\pi}{2}} \right| < 3.0 \times 10^{-9}$$

for the range

$$-1.0 \times 10^6 \leq x \leq 1.0 \times 10^6$$

EATAN

$$e \equiv \left| \frac{\text{atn}(x) - \text{atn}^*(x)}{\text{atn}(x)} \right| < 2.0 \times 10^{-9}$$

for the range

$$-3.88336148 \times 10^{37} \leq x \leq 3.88336148 \times 10^{37}$$

EEXP

$$e \equiv \left| \frac{e^x - (e^x)^*}{e^x} \right| < \left\{ \begin{array}{l} 2.0 \times 10^{-9} \; |x| \\ \text{or} \\ 2.0 \times 10^{-9} \end{array} \right\} \begin{array}{c} \text{whichever} \\ \text{is} \\ \text{greater} \end{array}$$

for the range

$$-\ln(\infty) < x < \ln(\infty)$$

$$\text{i.e., } 0 < e^x < \infty$$

ELN

$$e \equiv \left| \frac{\ln(x) - \ln^*(x)}{\ln(x)} \right| < 3.0 \times 10^{-9}$$

for the range

$$0 < x < \infty$$

ETANH

$$e \equiv \left| \tanh(x) - \tanh^*(x) \right| < 3.0 \times 10^{-9}$$

for the range

$$-\infty < x < \infty$$

ESQRT

$$e \equiv \left| \frac{\sqrt{x} - \sqrt{x}^*}{\sqrt{x}} \right| < 1.0 \times 10^{-9}$$

for the range

$$0 < x < \infty$$

STANDARD PRECISION SUBROUTINES

The following statements of accuracy apply to the standard precision subroutines.

FSIN

$$e \equiv \left| \frac{\sin(x) - \sin^*(x)}{x} \right| < 2.5 \times 10^{-7}$$

for the range

$$-1.0 \times 10^6 \leq x < 0$$

$$1.0 \times 10^6 \geq x > 0$$

for $x = 0$ $\sin(x) \equiv 0$

FCOS

$$e \equiv \left| \frac{\cos(x) - \cos^*(x)}{|x| + \frac{\pi}{2}} \right| < 2.5 \times 10^{-7}$$

for the range

$$-1.0 \times 10^6 \leq x \leq 1.0 \times 10^6$$

FATAN

$$e \equiv \left| \frac{\mathrm{atn}(x) - \mathrm{atn}^*(x)}{\mathrm{atn}(x)} \right| < 5.0 \times 10^{-7}$$

for the range

$$-3.883361 \times 10^{37} \leq x \leq 3.883361 \times 10^{37}$$

FEXP

$$e \equiv \left| \frac{e^x - (e^x)^*}{e^x} \right| < \left\{ \begin{array}{c} 2.5 \times 10^{-7} \, |x| \\ \text{or} \\ 2.5 \times 10^{-7} \end{array} \right\} \begin{array}{c} \text{whichever} \\ \text{is} \\ \text{greater} \end{array}$$

for the range

$$-\ln(\infty) < x < \ln(\infty) \text{ i.e., } 0 < e^x < \infty$$

FLN

$$e \equiv \left| \frac{\ln(x) - \ln^*(x)}{\ln(x)} \right| < 4.0 \times 10^{-7}$$

for the range

$$0 < x < \infty$$

FTANH

$$e \equiv \left| \tanh(x) - \tanh^*(x) \right| < 2.5 \times 10^{-7}$$

for the range

$$-\infty < x < +\infty$$

FSQRT

$$e \equiv \left| \frac{\sqrt{x} - \sqrt{x^*}}{\sqrt{x}} \right| < 2.5 \times 10^{-7}$$

for the range

$$0 < x < \infty$$

ELEMENTARY FUNCTION ALGORITHMS

The choice of an approximating algorithm for a given function depends on such considerations as expected execution time, storage requirements, and accuracy. For a given accuracy, and within reasonable limits, storage requirements vary inversely as the execution time. Polynomial approximating is used to evaluate the elementary functions to effect the desired balance between storage requirements and efficiency.

SINE-COSINE

Polynomial Approximation

Given a floating point number, x, n and y are defined such that

$$x\left(\frac{\pi}{2}\right) = n + y$$

where n is an integer and $0 \leq y < 1$. Thus, $x = 2\pi n + 2\pi y$, and the identities are

$$\sin x = \sin 2\pi y \text{ and } \cos x = 2\pi y .$$

The polynomial approximation, $F(z)$, for the function $(\sin 2\pi z)/z$ is used where $-1/4 \leq z \leq 1/4$.

The properties of sines and cosines are used to compute these functions as follows.

$$\cos 2\pi y = F(z)$$

where:

$z = 1/4 - y$ in the range $0 \leq y \leq 1/2$
$z = y - 3/4$ in the range $1/2 \leq y < 1$

$$\sin 2\pi y = F(z)$$

where:

$z = y$ in the range $0 \leq y < 1/4$
$z = 1/2 - y$ in the range $1/4 \leq y < 3/4$
$z = y - 1$ in the range $3/4 \leq y < 1$

Extended Precision

$$F(z) = z(a_1 + a_2 z^2 + a_3 z^4 + a_4 z^6 + a_5 z^8 + a_6 z^{10})$$

where

$a_1 = 6.2831853071$
$a_2 = -41.341702117$
$a_3 = 81.605226206$
$a_4 = -76.704281321$
$a_5 = 42.009805726$
$a_6 = -14.394135365$

Standard Precision

$$F(z) = a_1 z + a_2 z^3 + a_3 z^5 + a_4 z^7 + a_5 z^9$$

where:

$a_1 = 6.2831853$
$a_2 = -41.341681$
$a_3 = 81.602481$
$a_4 = -76.581285$
$a_5 = 39.760722$

ARCTANGENT

Polynomial Approximation

The routine for arctangent is built around a polynomial, F(z), that approximates Arctan(z) in the range $-.23 \leq z \leq .23$. The Arctan(z) for z outside this range is found by using the identities:

$$\text{Arctan}(-z) = -\text{Arctan}(z)$$

$$\text{Arctan}(z) = a_k + \text{Arctan}\left[\frac{z - b_k}{zb_k + 1}\right]$$

where

$$a_k = \frac{k\pi}{7} \text{ and } b_k = \tan a_k$$

and k is determined so that

$$\tan\frac{(2k-1)\pi}{14} \leq z < \tan\frac{(2k+1)\pi}{14} \quad k = 1, 2, 3$$

Having determined the value of k appropriate to z, the transformation $x = (z-b_k)/(zb_k + 1)$ puts x in the range $-\tan \pi/14 \leq x < \tan \pi/14$. The polynomial F(z) was chosen to be good over a range slightly larger (i.e., $.23 > \tan \pi/14$) so that the comparisons to determine the interval in which z lies need be only standard precision accuracy.

Extended Precision

$$F(z) = x(1.0 - a_1 x^2 + a_2 x^4 - a_3 x^6 + a_4 x^8)$$

$a_1 = .33333327142$
$a_2 = .19999056792$
$a_3 = .14235177463$
$a_4 = .09992331248$

Standard Precision

$$F(z) = x\,(1.0 - .333329573z^2 + .199641035z^4 - .131779888z^6)$$

SQUARE ROOT

Square Root (x)
Let $x = 2^{2b}F$ when $.25 \le F < 1$
then $\sqrt{X} = 2^b\sqrt{F}$
where $\sqrt{F} = P_i$ i = number of approximation

$P_1 = AF + B$ as a first approximation followed by 2 Newton iterations

where

$A = .875,\ B = .27863$ when $.25 \le F < .5$

or

$A = .578125,\ B = .421875$ when $.5 \le F < 1$

$$P_2 = \frac{\left(P_1 + \frac{F}{P_1}\right)}{2}$$

$$P_3 = \frac{\left(P_2 + \frac{F}{P_2}\right)}{2}$$

NATURAL LOGARITHM

Polynomial Approximation

Given a normalized floating point number

$$x = 2^k \times f\left(\frac{1}{2} \le f < 1\right),$$

j and g are found such that $x = 2^j g$ where $(\sqrt{2}/2 \le g < \sqrt{2})$. This is done by setting $j = k-1$, $g = 2f$ if $f < \sqrt{2}/2$ and $j = k$, $g = f$ otherwise.

Thus:

$$\ln(x) = j \cdot \ln(2) + \ln(g).$$

The approximation for $\ln(g)$, $\sqrt{2}/2 \le g < \sqrt{2}$, is based on the series

$$\ln\frac{v+x}{v-x} = 2\left[(x/v) + (x^3/3v^3) + (x^5/5v^5) + \ldots\right]$$

which converges for $(-v < x < v)$.
With the transformation

$$x = v\frac{f-1}{f+1}, \quad v = (\sqrt{2}+1)^2$$

so that $-1 \le x < 1$ for $\sqrt{2}/2 \le g < \sqrt{2}$.
Substituting,

$$\ln(g) = 2\,(z + z^3/3 + z^5/5 + \ldots)$$

where $z = x/v = (f-1)/(f+1)$. The approximation used is $G(z)$ for $\ln(g)/z$ in the range $\sqrt{2}/2 \le g < \sqrt{2}$.

Extended Precision

$$G(z) = b_0 + b_2z^2 + b_4z^4 + b_6z^6 + b_8z^8$$

$b_0 = 2.0$
$b_2 = .666666564181$
$b_4 = .400018840613$
$b_6 = .28453572660$
$b_8 = .125$
$z = \frac{g-1}{g+1}$
$\sqrt{2}/2 = .7071067811865$
$\ln(2) = .6931471805599$

Thus, the required calculation is:

$$\ln(x) = j \cdot \ln(2) + zG(z)$$

Standard Precision

$$G(z) = 2.0 + .66664413786\,z^2 + .4019234697z^4 + .25z^6$$

EXPONENTIAL

Polynomial Approximation

To find e^x, the following identity is used.

$$e^x = 2^{\left(x \log_2 e\right)}$$

To reduce the range, we let

$$x \log_2 e = n + d + z$$

where:

- n is the integral portion of the real number,
- d is a discreet fraction (1/8, 3/8, 5/8, or 7/8) of the real number, and
- z is the remainder which is in the range $-1/8 \leq z \leq 1/8$.

Thus,

$$e^x = 2^n \times 2^d \times 2^z$$

and it is necessary to only approximate 2^z for $-1/8 \leq z \leq 1/8$ by using the polynomial $F(z)$.

Extended Precision

$$F(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4 + a_5 z^5$$

where:

a_0 = 1.0
a_1 = .69314718057
a_2 = .24022648580
a_3 = .055504105406
a_4 = .0096217398747
a_5 = .0013337729375

Standard Precision

$$F(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + a_4 z^4$$

where:

a_0 = 1.0
a_1 = .693147079
a_2 = .240226486
a_3 = .0555301557
a_4 = .00962173985

HYPERBOLIC TANGENT

$$\text{Tanh}(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

for

$x \geq 32$	Tanh (x) = 1
$x \leq -32$	Tanh (x) = -1

FLOATING-POINT BASE TO AN INTEGER EXPONENT

$$A = e^{\ln A}$$

therefore:

$$A^B = \left(e^{\ln A}\right)^B = e^{B \ln A}$$

SELECTIVE DUMP AND MISCELLANEOUS SUBROUTINES

These subroutines allow the user to dump or trace selected portions of core storage at object time and also allow certain machine functions to be performed using the FORTRAN language.

SELECTIVE DUMP SUBROUTINES

These subroutines allow the user to dump selected portions of core storage during execution of an object program. There are two dump subroutines: one to dump selected data on the printer assigned as the LIST printer; and one to dump status information on that printer. These dump routines can only be called from a single interrupt level or from the mainline level while debugging a core load (i.e., they are not re-entrant).

Dump Selected Data

A subroutine is available for the purpose of selecting an area of core storage and having it dumped out on the LIST printer. This subroutine has two entry points, one for hexadecimal output, and one for decimal output. The entry points for the various configurations are shown below:

Entry Point	Function of Subroutine
DMPHX	Dump on the LIST printer, using hexadecimal output.
DMPDC	Dump on the LIST printer, using decimal output.

Calling Sequence

The calling sequence for both of the above functions is as follows:

```
    CALL        ENTRY POINT
    DC          START
    DC          END
```

START and END represent the starting and ending addresses of the portion of core storage to be dumped. The starting and ending addresses may be the same, in which case, one word will be dumped.

Format

Before the actual dump appears on the selected output device, the user is given one line of status information. This line indicates the status of the overflow and carry triggers, and the contents of the A- and Q-registers and the three index registers. The register contents are given in the same form (hexadecimal or decimal) as the rest of the dump. The format of the status information is shown below:

$$\underbrace{0,1,2,\text{ or }3}_{\text{Status}} \quad \underbrace{\text{HHHH}(\pm\text{ DDDDD})}_{\text{A-Register}} \quad \underbrace{\text{HHHH}(\pm\text{ DDDDD})}_{\text{Q-Register}}$$

$$\underbrace{\text{HHHH}(\pm\text{ DDDDD})}_{\text{Index Register 1}} \quad \underbrace{\text{HHHH}(\pm\text{DDDDD})}_{\text{Index Register 2}} \quad \underbrace{\text{HHHH}(\pm\text{ DDDDD})}_{\text{Index Register 3}}$$

The status will be zero if both overflow and carry are off, one if only overflow is on, two if only carry is on, or three if both overflow and carry are on.

All other data is dumped eight words to a line, with the address of the first word in each line printed to the left of the line. Hexadecimal data is printed four characters per word; decimal data is printed five digits per word with a preceding plus or minus sign.

Page numbers will not be printed for either subroutine.

The dump selected data routine can be called from a FORTRAN program using the statement:

```
    CALL DMP (I, J, K)
```

where

I is a control parameter that specifies the form of the dump that will be made on the list printer.

0 - indicates hexadecimal form

1 - indicates decimal form

J is an integer variable that contains the start address to be dumped.

K is an integer variable that contains the end address to be dumped.

Dump Status

This subroutine provides a relatively easy and efficient means of dumping the status of the 1800. Words that contain status information relating to index registers, status indicators, the Q-register, and the contents of a portion of the work area in use are dumped. The number of the level is also printed. This information may frequently be required when testing a program.

This subroutine is called via the following statement:

```
CALL DMPST
DC (no. wds of wk area)
```

The words of core storage are dumped on the list printer in hexadecimal form with a space between each word. The dump status routine can be called from a FORTRAN program using the statement:

```
CALL DMPS (I)
```

where

I is an integer variable that specifies the number of words of the work area to be printed. The DMPS routine calls the DMPST routine, which uses the list printer for output.

TRACE INTERRUPT ROUTINES

A user-written mainline trace interrupt routine can be included in a process core load to process a trace interrupt. The routine can be designed to monitor a number of conditions such as checking all branch instructions, checking all instructions, except branch, checking all instructions within a defined limit, or a combination of these and many other options.

The following items should be noted in writing a trace interrupt routine.

1. The trace interrupt routine should be written as a CALL or LIBF subroutine.
2. The index registers, A- and Q-registers, and Carry and Overflow status that are used by the routine should be saved and restored.
3. The address of the current instruction is available indirectly via absolute location 9.
4. The subroutine must be specified on an *INCLD control card at core load build time.
5. The conversion subroutine CONHX and the print subroutine TRPRT are used for listing the trace output. These are the only IBM-supplied subroutines that can be called from the user-written trace routine.

Trace Print

The trace print subroutine (TRPRT) is designed to be called by an assembler-language trace interrupt subroutine. It prints specified characters on the 1053/1816 or 1443 printers without masking any interrupts. Indicators are sensed to determine the status of the selected unit. The calling sequence is as follows.

```
CALL    TRPRT
DC      n
DC      OUTPT
```

where

n = 0 for the 1053/1816 printer and n = 1 for the 1443 printer. The first printer of the first group is used for output when a 1053 or 1816 is specified. No backup is available.

OUTPT specifies the number of words to be printed. The output area begins at OUTPT + 1.

TRPRT saves and

OUTPT specifies the number of words to be printed. The output area begins at OUTPT + 1.

TRPRT saves and restores the A- and Q-registers, XR1 and XR2, and the status of the carry and overflow indicators (XR3 is not used). If an 1816 is specified, the keyboard indicator status is saved and then set busy. The indicator is restored to its original status when TRPRT exits to the calling program.

Trace Hexadecimal Conversion

The trace hexadecimal conversion subroutine (CONHX) is designed to be called by an assembler-language trace interrupt subroutine. It converts hexadecimal data to printer codes. The digits to be converted are placed, four at a time, in the A-register and the following calling sequence is used.

```
CALL    CONHX
DC      n
DC      OUTPT
```

where

n = 0 for conversion to the 1053/1816 printer code, and n = 1 for conversion to the 1443 printer code.
OUTPT is the address of the first of two consecutive locations where the printer code characters are to be placed (two characters per word).
CONHX does not mask interrupts, but does save and restore the Q-register, XR1, XR2, and XR3. No re-entrant provisions are made.

OVERLAY ROUTINE (FLIP)

The 1800 TSX subroutine library contains a flipper routine which is used to call LOCAL (load on call) routines into core storage. FLIP passes the total word count to DISKN and that routine reads in the entire LOCAL. When a LOCAL routine is called, control is passed to the flipper routine which reads the LOCAL into core storage and transfers control to it. All LOCALs in a given core load are executed from the same core storage locations; each LOCAL group overlays the previous one.

MACHINE FUNCTION SUBROUTINES

These subroutines allow the user to perform certain machine functions using the FORTRAN language.

OR Function

This subroutine performs an OR operation on the two parameters specified. The function must be specified as:

```
IOR (I,J)
```

where

I and J are integer expressions.

The values of I and J are logically ORed and the result is placed in the accumulator; I and J are not changed.

Example:

```
   .
   .
K = IOR (I,J)
   .
   .
```

I and J are logically ORed and the result is stored in K.

The following table shows the result of an OR operation on individual bits.

I	1	1	0	0
J	1	0	1	0
Result	1	1	1	0

Exclusive OR Function

This subroutine performs an Exclusive OR operation on the two parameters specified. The function must be specified as:

```
IEOR (I,J)
```

where

I and J are integer expressions.

The values of I and J are Exclusively ORed and the result is placed in the accumulator; I and J are not changed.

Example:

```
   .
   .
K = IEOR (I,J)
   .
   .
```

I and J are exclusively ORed and the result is stored in K.

The following table shows the result of an Exclusive OR operation on individual bits.

I	1	1	0	0
J	1	0	1	0
Result	0	1	1	0

AND Function

This subroutine performs an AND operation on the two parameters specified. The function must be specified as:

```
IAND (I,J)
```

where: I and J are integer constants or integer variables.

The contents of I and J are logically ANDed and the result is placed in the accumulator; I and J are not changed.

Example:

```
   .
   .
   .
IF (IAND (I,J)) 3, 2, 5
```

I and J are ANDed and the result is tested in the IF statement. There will be: a transfer to statement 3 if the AND result is negative; a transfer to statement 2 if the result is zero; or a transfer to statement 5 if the result is positive.

The following table shows the result of an AND operation on individual bits.

I	1	1	0	0
J	1	0	1	0
Result	1	0	0	0

Load Function

This subroutine performs a load operation for the parameter specified. The function must be specified as:

```
LD (I)
```

where: I is an integer expression that specifies a core storage address. The contents of the core storage are moved to the accumulator. This allows "testing for busy", etc., of known storage locations outside of the program area.

The core storage locations for the various timers maintained by the ITC program are given in Table 8. For various examples of the LD subroutine see Interval Timer Control Program.

Call Mask

The mask subroutine is used to perform a mask operation which prevents interrupt recognition on the designated levels. The status of levels not designated is unchanged. The format of this statement is:

```
CALL MASK (I, J)
```

where: I and J are integer expressions which designate the level(s) to be masked. I refers to levels 0 through 13, J refers to levels 14 through 23. Both parameters are always required.

Table 8. Timer Locations

Name	Core Storage Location
Machine Timers	
A	00004
B	00005
C	00006
Programmed Timers	
1	00062
2	00065
3	00068
4	00071
5	00074
6	00077
7	00080
8	00083
9	00086
Time-Sharing Clock	00089

Example:

0-13
14-23

Conditions:

DATA statements may be used in conjunction with the CALL MASK (and CALL UNMK) statement (see the publication, IBM 1800 FORTRAN Language, Form C26-5905). They can be set up as follows.

Problem:

Mask levels 5, 7, 11, 12, 21, 22, and 23.

Solution:

```
DATA I, J/Z0518,Z01C0/
CALL MASK (I, J)
```

Call Unmask

The unmask subroutine is used to perform an unmask operation which allows interrupts to be recognized on the designated levels. The status of levels not designated is unchanged. The format of the statement is:

```
CALL UNMK (I, J)
```

where I and J are integer expressions which designate the levels to be unmasked. I refers to levels 0 through 13, J refers to levels 14 through 23. Both parameters are always required.

Example:

Conditions:

Same as for CALL MASK in previous example.

Problem:

5 7 11 12 21 22 23

Unmask levels 1, 2, 3, 5, 12, and 21.

Solution:

```
DATA I, J/Z7408, Z0100/
CALL UNMK (I, J)
```

NOTES: 1. The combination of the examples (in the sequence given) for the CALL MASK and CALL UNMK statements would produce the following conditions.

Levels 1, 2, 3, 5, 12, and 21 are unmasked.

Levels 7, 11, 22, and 23 are masked.

Levels 4, 6, 8, 9, 10, 13-20 are unchanged.

2. The mask and unmask subroutines maintain a current record of the interrupt level mask status. This is necessary since the system director sometimes masks all levels and then restores the status according to this record. Thus, the user should always mask and unmask via these routines in order to keep this record current.

Call Save Mask

The save mask subroutine causes the contents of the current mask words (conditions set by previous CALL MASK or CALL UNMK statements) to be moved to variables so that they can be restored at a later time, if desired. The statement format is:

```
CALL SAVMK (I, J)
```

where I and J are integer variables that will receive the contents of the retained mask words.

Call Restore Mask

The restore mask subroutine is used to perform a mask and unmask operation to restore the interrupt mask register to its previously saved condition. The variables used as parameters are normally those named in a previous CALL SAVMK statement. The format of the statement is:

```
CALL RESMK (I, J)
```

where I and J are integer expressions which designate the levels to be masked. I refers to levels 0 through 13, J refers to levels 14 through 23. Both parameters are always required.

Example:

Conditions:

Same as for CALL MASK in previous example.

Problem:

Mask levels 5, 7, 9, 10, and 12.

Unmask all other levels.

Solution:

```
DATA I, J/Z0568,Z0/
CALL RESMK (I, J)
```

Call Operation Monitor

This subroutine is used to reset the operation monitor. Once the operation monitor has been activated by the operator, the reset command must be executed frequently enough to prevent the timer from timing out. The format of the statement is:

```
CALL OPMON
```

WRITING ASSEMBLER LANGUAGE SUBROUTINES

User-written assembler language subroutines must follow the writing specifications outlined below.

The subroutine source statements shown in the following examples should be preceded and followed by the following control cards for the assembly process.

```
// JOB
// ASM
*LIST
*PRINT SYMBOL TABLE
    .
    .
    .
SUBROUTINE SOURCE DECK
    .
    .
    .
// DUP
*STORE          NAME
```

CALL Subroutines

A subroutine that is called by a CALL statement is linked to via a long BSI instruction. For example, a FORTRAN source statement

```
CALL SUB (I, J, K, 101)
```

or an assembler language calling sequence

```
CALL      SUB
DC        ADDRI
DC        ADDRJ
DC        ADDRK
DC        ADCON
```

appears in core at execution time as

```
BSI   L   SUB
DC        ADDRI
DC        ADDRJ
DC        ADDRK
DC        ADCON
```

where ADDRI, ADDRJ and ADDRK are the core addresses at which the variables I, J, and K are stored, and ADCON is the core address where the constant, 101, is stored.

Note that most subroutines entered by an assembler language calling sequence expect the constants themselves to appear in the calling sequence rather than the address of the constants. Therefore, not all subroutines entered by a CALL can be called from a FORTRAN program.

The following example illustrates how to define the entry point, save the contents of the registers, get the parameters, and return to the calling program.

```
      HDNG      SAMPLE CALL SUBROUTINE
      ENT       SUB    DEFINES ENTRY POINT
SUB   DC        0      SUBROUTINE ENTRY POINT
      STD       TEMP   SAVE A AND Q REGISTERS
      STX   1   XR1+1  SAVE INDEX REGISTERS
      STX   2   XR2+1
      STX   3   XR3+1
      LDX   I1  SUB    SET XR1 TO PARAMETER
      LD    I1  0      GET FIRST PARAMETER
      STO       PARA1
      LD    I1  1      GET SECOND PARAMETER
      STO       PARA2
      LD    I1  2      GET THIRD PARAMETER
      STO       PARA3
```

```
         MDX   1   3       SET UP TO RETURN TO MAINLINE
         STX   1   SUB     FOLLOWING THIRD PARAMETER
         LDX   I3  103     SET XR3 TO TRANSFER VECTOR IF
*                          ANY LIBF ARE TO BE MADE WITHIN
*                          THE PROGRAM
*
*                          EXECUTION INSTRUCTIONS
*USE     STO   I1  0       TO STORE RESULT IN I, ETC.
*USE     STO   L   /FFFF   TO STORE RESULT IN FIRST
*                          WORD OF FORTRAN COMMON.
*USE     LIBF      FLD     (OR ELD) AND
*        DC        DATA    TO STORE RESULT IN FAC.
*
*                          EXIT FROM SUBROUTINE
XR1      LDX   L1  *-*     RELOAD INDEX REGISTERS
XR2      LDX   L2  *-*
XR3      LDX   L3  *-*
         LDD       TEMP    RELOAD A AND Q REGISTERS
         BSC   I   SUB     RETURN TO MAINLINE
TEMP     BSS   E   2
PARA1    DC        0
PARA2    DC        0
PARA3    DC        0
         END
```

LIBF Subroutines

The source statements for subroutines that are called by a LIBF statement must be preceded by a LIBR statement.

At execution time, the LIBF call appears as a BSI instruction indexed by XR3 and with a displacement that reflects the transfer vector entry for the subroutine being called. XR3 contains the address of the transfer vector. The transfer vector entry contains a long BSI instruction to the subroutine entry point.

The following example illustrates a LIBF subroutine and shows how to define the entry point, save the machine status, get the address of the parameter list, and return to the calling program.

```
         HDNG      SAMPLE LIBF SUBROUTINE
*        LIBF      SUB1    SOURCE LANGUAGE
*        DC        PARA1   CALLING SEQUENCE
         LIBR              LIBF CONTROL CARD
         ENT       SUB1    DEFINES ENTRY POINT
SUB1     DC        0       SUB1 ENTRY POINT
         STD       TEMP    SAVE MACHINE STATUS
         STX       X1+1    THAT IS TO BE USED
         LDX   I1  SUB1    GET ADDRESS OF PARA LIST
         LD    1   -3      FROM TV
         STO       *+1
         LDX   L1  *-*     XR1 POINTS TO PARA LIST
         LD    1   0       GET PARAMETER
         STO       PARA1
         MDX   1   1       SET UP RETURN ADDR IN SUB1
         STX   1   SUB1
*SUBROUTINE OPERATION
X1       LDX   L1  *-*     RESTORE MACHINE
         LDD       TEMP
         BSC   I   SUB1    RETURN TO MAINLINE
TEMP     BSS   E   2
PARA1    DC        0
         END
```

Input/Output Subroutines

The procedures for writing input/output subroutines are similar to those for CALL or LIBF subroutines, except that an ISS statement is used to define the entry of the call section of the routine; also, the interrupt entry points must be defined.

The basic identification for the interrupt entry portion is the IAC code. There is a unique IAC code for each ILSW bit that is turned on by an I/O interrupt. At system generation time, the user defines the IAC codes and their corresponding ILSW bit. The same IAC code must be used when writing an I/O subroutine.

As stated previously, an ISS statement is used to define the call entry point (only one call entry point is permitted). If the subroutine is to be called by a LIBF statement, the ISS statement must be preceded by a LIBR statement. The LIBR statement is omitted if the subroutine is to be called by using a CALL statement (the CALL statement method must be used if the subroutine is to be called from a FORTRAN program). Following the ISS statement, there must be a pair of DC statements for each interrupt entry point. The first DC statement must define the IAC code for that entry, and the second DC must define the address of the interrupt entry point. This is followed by an ORG*-2X where X is the number of pairs of DC statements.

```
         HDNG      SAMPLE CARD I/O ROUTINE
*                         LOADER INFORMATION
         LIBR             SIGNIFIES THIS IS A LIBF
         ISS   2   CARD   2= NO. OF INT. ENTRY POINTS
*                  CARD IS LIBF ENTRY POINT
         DC        2      1442-1 IAC CODE
         DC        INT1   1442-1 INTERRUPT ENTRY POINT
         DC        17     1442-2 IAC CODE
         DC        INT2   1442-2 INTERRUPT ENTRY POINT
         ORG       *-4    CAUSES OVERLAY OF LDER INFO
*                         DESCRIPTION OF CALL
*        LIBF      CARD
*        DC        /A00B  A=0 READ, A=1 PUNCH, B=0 1442-1
*        DC        AREA   I/O AREA
CARD     DC        0      LIBF ENTRY POINT
         BSI   I   172    CALL TVSAVE TO SAVE MACHINE
*                         REGISTERS AND STATUS. ALSO
*                         SETS WORD 55 TO POINT TO FIRST
*                         PARAMETER (RETURN ADDRESS)
         LDX   I1  55     XR1=LIBF PARAMETERS
         LDX   L2  CD1    XR2=1442-1 DEVICE TABLE
         LD    X1  O      DETERMINE DEVICE
         BSC       E      SKIP IF FIRST 1442
         MDX   2   CD2-CD1 INCREMENT TO POINT TO 1442-2
         SRA       12     TEST FOR READ OR PUNCH
         BSC   L   PUNCH,Z BRANCH IF FUNCTION IS PUNCH
```

```
READ    LD    X1  1       GET SECOND PARAMETER
        STO   X2  RD      STORE IN IOCC
        XIO   X2  SENSE   SENSE DSW
        BSC   L   *-3,E   LOOP IF 1442 IS NOT READY
        XIO   X2  RD      READ  A  CARD
        MDX   L   7,1     INCREMENT GENERAL I/O BUSY IND
        NOP
        MDX       OUT     EXIT BACK TO USER VIA OUT.
PUNCH   LD    X1  1       GET SECOND PARAMETER
        STO   X2  PH
        XIO   X2  SENSE   SENSE DSW
        BSC   L   *-3,E   LOOP IF 1442 IS NOT READY
        XIO   X2  PH      PUNCH A CARD
        MDX   L   7,1     INCREMENT GENERAL I/O BUSY IND
        NOP
OUT     MDX    1  2       XR1=RETURN ADDRESS TO USER
        STX   L2  55      55 NOW CONTAINS RETURN ADDRESS
        BSI   I   173     RETURN TO USER VIA TV EXIT
*                         INTERRUPT ROUTINE
INT1    LDX   L2  CD1     XR2=1442-1.  THIS IS INTERRUPT
*                         ENTRY FOR 1442-1
        MDX       *+2
INT2    LDX   L2  CD2     XR2=1442-2. INTERRUPT ENTRY
*                         FOR 1442-2.
        XIO   X2  SENSR   SENSE DSW WITH RESET
        SLA       2       TEST FOR ERROR
        BSC       +Z      SKIP  IF NO ERROR
        MDX       *-1     HALT IF ERROR
        MDX   L   7,-1    DECREMENT GEN I/O BUSY IND.
        NOP
        BSC   I   90      RETURN TO MIC
*                         DEVICE TABLES
        BSS   E   0       IOCC MUST START AT EVEN ADDRESS
CD1     DC        0       DEVICE TABLE FOR 1442-1
        DC        /1600   READ IOCC
        DC        0
        DC        /1500   PUNCH IOCC
        DC        0
        DC        /1700   SENSE IOCC
        DC        0
        DC        /1701   SENSE/RESET IOCC
CD2     DC        0       DEVICE TABLE FOR 1442-2
        DC        /8E00   READ IOCC
        DC        0
        DC        /8D00   PUNCH IOCC
        DC        0
        DC        /8F00   SENSE IOCC
        DC        0
        DC        /8F01   SENSE/RESET IOCC
*                         DEVICE TABLES EQUATES
RD      EQU       0       READ IOCC
PH      EQU       2       PUNCH IOCC
SENSE   EQU       4       SENSE IOCC
SENSR   EQU       6       SENSE/RESET IOCC
        END
```

PROGRAMMING MULTI-LEVEL INTERRUPTS BY USING RE-ENTRANT CODING

Need For Re-Entrant Coding

One of the basic problems that arises in multi-level programming is requirement of the same subroutine by different levels of operation.

For example, the computer is servicing a main-line program which is executing a square-root subroutine when an external interrupt occurs. The hardware interrupt will automatically branch to an address which will allow servicing of the interrupt.

The program that services the interrupt may also require use of a square-root subroutine. If a method of re-entrant coding were not used, the identical square-root subroutine would have to be in core storage twice (once for each program that called it); otherwise, the intermediate results which are needed when the computer returns to complete the mainline program would be destroyed by the interrupt program.

Concept Of Level Work Areas

To allow one subroutine to be entered at any time and from any interrupt level, without loss of intermediate results, a method of re-entrant coding using level work areas is used.

(Re-entrant coding is defined as coding which allows a program to be entered and executed from different levels without destroying the intermediate results.)

The IBM 1800 TSX System provides features which facilitate the coding of re-entrant subroutines.

Each interrupt level specified by the user is provided with a level work area of 100 locations, which are reserved for the exclusive use of programs operating on that priority level.

The first 58 of these locations are specifically reserved for use by the TSX System, while the remaining 42 locations are available to allow all other subroutines to maintain the ability to re-enter.

The start address of the level work area for any priority level always appears in location LWA (fixed location $104_{10} = 68_{16}$). If an index register is loaded with the contents of this location, and all references to temporary storage locations are indexed, 42 temporary storage locations are made available to the subroutine for each level it may be operating on. If the subroutine is re-entered, different effective addresses are generated for each such indexed operand, and the re-entry problem is solved.

The following sequence of instructions illustrates how the contents of the A-register are saved in TEMP in the level work area and later restored by the instruction at LOAD:

```
LWA    EQU        104
TEMP   EQU        58

       LDX   I1   LWA    WORK AREA ADDR TO XR1
STRE   STO   1    TEMP   SAVE A-REGISTER IN TEMP
        .
        .
        .
LOAD   LD    1    TEMP   RESTORE A-REGISTER
```

Should the subroutine be interrupted and re-entered, there will be no storage conflicts, since the contents of LWA changes with each interrupt level. Hence, the instructions at STRE and LOAD reference different effective addresses for each interrupt level.

Mechanism For Re-Entrant Control

For each interrupt serviced, MIC (Master Interrupt Control program) saves and subsequently restores the contents of the A- and Q-registers, index registers, machine status, and locations WK4 ($54_{10} = 36_{16}$) and WK5 ($55_{10} = 37_{16}$). MIC also sets LWA to the correct level work area address for each interrupt level.

Since locations WK4 and WK5 are saved by MIC for each interrupt level, these locations may also be used for temporary storage by re-entrant subroutines, e.g., loading and storing of index registers. Furthermore, these locations are also used for other purposes, as explained below.

Protecting Entry and Return Addresses

The first location of a callable subroutine is set by a BSI instruction. As with all fixed locations upon re-entry, this location may be changed and the return address may be lost. The TSX System supplies two pairs of subroutines which provide a method of protecting the return address. They also perform several additional functions useful for subroutines.

Subroutines Referenced by a CALL Instruction

For subroutines referenced by a CALL (2-word BSI) instruction:

```
QZSAV  EQU        154
QZEXT  EQU        155

SUBRT  DC         0      ENTRY TO CALL ROUTINE
       BSI   I    QZSAV  CALL TO QZSAV
        .
        .
        .
EXIT   BSI   I    QZEXT  EXIT FROM SUBROUTINE
```

The QZSAV subroutine saves the contents of index registers 1, 2, and 3, the A- and Q-registers and machine status and places the return address in location WK4 ($54_{10} = 36_{16}$). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to $127_{10} = 7F_{16}$.

The QZEXT subroutine restores the index registers, machine status, and A- and Q-registers and returns control to the calling routine via a BSI I WK4. The address set in WK4 by QZSAV must, therefore, be incremented by 1 for every parameter following the CALL.

Subroutines Referenced By A LIBF Instruction

For subroutines referenced by a LIBF (1-word BSI) instruction through the transfer vector:

```
TVSAV  EQU        172
TVEXT  EQU        173

SUBRT  DC         0      ENTRY TO LIBF ROUTINE
       BSI   I    TVSAV  CALL TO TVSAV

EXIT   BSI   I    TVEXT  EXIT FROM SUBROUTINE
```

The TVSAV subroutine saves the contents of index registers 1 and 2, the A- and Q-registers, and machine status and places the return address in WK5 ($55_{10} = 37_{16}$). In addition, index registers 1 and 3 are set to the first location of the level work area, and index register 2 is set to $127_{10} = 7F_{16}$.

The TVEXT subroutine restores the contents of index registers 1 and 2, A- and Q-registers, and machine status. Index register 3 is set to the transfer vector location, and control is returned to the calling routine via a BSI I WK5. The address set in WK5 by TVSAV must, therefore, be incremented by 1 for every parameter following the LIBF.

Other Considerations

It should be understood that the use of QZSAV or TVSAV does not obviate careful logic control. If parameters follow the call to the subroutine, it is the responsibility of the subroutine to obtain these parameters and to adjust WK4 or WK5.

If subroutines are nested, that is, one subroutine calls another, care must be exercised to save and restore the storage locations used by QZSAV and TVSAV across the nested call, as well as the return address in WK4 or WK5. Furthermore, nested subroutines must be planned so that the same locations in the level work area are not used by more than one subroutine.

Note that TVSAV, TVEXT, QZSAV, and QZEXT are referenced by indirect BSI instructions and not by CALL statements. The call to TVSAV or QZSAV must be the first instruction executed in (and immediately following) the entry location, as illustrated.

Masking Out The Interrupts

Another method of providing re-entrant coding is to prevent the interrupts from being recognized. This may be accomplished through appropriate use of the TSX subroutines MASK, SAVMK, RESMK and UNMK.

In assembler language, it is possible to use the XIO command with the IOCC-Masking words provided by the TSX system. To mask all interrupt levels completely, the following instructions may be executed:

```
MSK1   EQU        50     LOCATIONS OF MASK IOCC WORDS
MSK2   EQU        52

       XIO   L    MSK1   MASK ALL INTERRUPT LEVELS
       XIO   L    MSK2
```

To restore the interrupt mask status, the following instructions may be executed:

```
       XIO   L    MSK3   RESTORE INTERRUPT STATUS
       XIO   L    MSK4
MSK3   EQU        46     LOCATIONS OF UNMASK IOCC WORDS
MSK4   EQU        48
```

This particular method of re-entrant coding is effective and permissible, but is, in general, undesirable. If interrupts are prevented from being recognized as may occur, the philosophy of the IBM 1800 interrupt system is defeated. However, for short sequences of instructions, the method of masking out the interrupts may be the fastest means of obtaining re-entrant coding.

Programming Notes

An illustration of the use of WK4, WK5, and masking occurs when an index register is to be loaded with a value from the level work area. Double indexing can not be done; therefore, it is impossible with one instruction to load an index register with a value that can be reached only by an index register. The following examples illustrate re-entrant and non-re-entrant methods of loading an index register.

Non-re-entrant

```
CONST   EQU          53
         .
         .
         .
        LD     1     CONST
        STO          * + 1
        LDX    L2    * - *
```

Re-entrant

```
CONST   EQU          53
MSK1    EQU          50
MSK2    EQU          52
MSK3    EQU          46
MSK4    EQU          48
         .
         .
         .
        XIO    L     MSK1
        XIO    L     MSK2
        LD     1     CONST
        STO          * + 1
        LDX    L2    * - *
        XIO    L     MSK3
        XIO    L     MSK4
```

or

```
CONST   EQU          53
         .
         .
         .
        LD     1     CONST
        STO    L     54        (location WK4)
        LDX    I2    54
```

This section describes the Assembler language statements to be used in place of the FORTRAN CALL statements provided in the time-sharing executive system.

Machine Interval Timers

The assembler language statements to call the TIMER subprogram are:

```
CALL  TIMER
CALL  NAME
DC    A
DC    B
```

where NAME is the name of the subprogram to be executed when the time specified by B has elapsed. A and B must be defined as:

A	DC	1	for machine interval timer (A).
		or	
		2	for machine interval timer (B).
B	DC	XX	the number of intervals to be counted before the subprogram is executed.

Programmed Interval Timers

The assembler language statements to call the COUNT subprogram are:

```
CALL  COUNT
DC    A
DC    B
DC    C
```

where the parameters A, B, and C must be defined as:

B	DC	1 through 9	programmed timer number
C	DC	XX	the number of intervals to be counted before the subprogram is executed.
A	DC	0 through 31	number of the subprogram to be executed when the time has elapsed.

The assembler language statements to be used to read and to set the programmed real-time clock are:

Read:

```
CALL  CLOCK
DC    A
```

where A is the address of the location where the contents of the clock are to be stored.

Set:

```
CALL  SETCL
DC    A
```

where A must be defined as:

A	DC	XXXX	the time to be used for setting the clock. The time must be represented in hours and thousandths of hours (i.e., 00000 through 23999).

PSC Statements:

The following assembler language statements are equivalent to the FORTRAN language calls for core load sequencing.

```
CALL BACK     no parameters are required for
              these calls.
CALL ENDTS
CALL VIAQ
CALL DPART
```

Call Chain:

```
CALL CHAIN
CALL NAME
```

where NAME is the name of the core load to be executed.

Call Special:

```
CALL SPECL
CALL NAME
```

where NAME is the name of the core load to be executed.

Call Queue:

```
CALL QUEUE
CALL NAME
DC    A
DC    B
```

where NAME is the name of a core load to be added to the queue. A and B must be defined as follows.

A	DC	1 through 32,767	priority number.
B	DC	1-32,766	replace lowest priority entry.
		or	
		0	ignore the call.
		32,767	restart.

Call Unqueue:

```
CALL UNQ
CALL NAME
DC    A
```

where NAME is the name of a core load whose entry is to be removed from the queue. A must be defined as follows.

A	DC	1 through 32,767	~~priority number~~ priority number

Call Time-Share:

```
CALL SHARE
DC    A
```

where A must be defined as follows:

A	DC	XX	number of programmed timer base intervals to be used for nonprocess operations.

Call Programmed Settable Interrupts:

```
CALL LEVEL
DC    A
```

where A must be defined as:

A	DC	0 to 23	user specified hardware level to cause interrupt.

Interrupt Calls:

The following assembler language statements are used to service and clear recorded interrupts.

Call Interrupt Exit:

```
CALL INTEX no parameters are required for
           this call.
```

Service Recorded Interrupts:

```
CALL QIFON
CALL NAME
DC    A
DC    B
DC    C
DC    D
```

where NAME is the name of the core load to be serviced if recorded. A, B, C, and D must be defined as follows:

A	DC	XX	priority number.
B	DC	XX	interrupt level number or indicator.
C	DC	XX	position within PISW or indicator.
D	DC	1-32,766	replace lowest priority entry. 1-32766
		or	
		0	ignore the call.
		32,767	restart

Clear Recorded Interrupts:

```
CALL CLEAR          CALL CLEAR
DC    A             DC   A (when A = 0)
DC    B(1)
DC    C(1)
DC    B(2)
DC    C(2)
```

where A, B, and C must be defined as follows:

```
A    DC    XX  number of Bs and Cs
               which follow. If zero,
               all recorded status is
               cleared.
B    DC    XX  interrupt level number
               or indicator. Not used
               if A = 0.
C    DC    XX  position within PISW or
               indicator. Not used
               if A = 0.
```

Miscellaneous Subroutines:

The following assembler language statements are used to link the miscellaneous subroutines.

Mask:

```
CALL  MASK
DC    A
DC    B
```

where A and B must be defined as:

```
A    DC    /0000  levels to masked. A
                  represents the first 14
                  levels (0 through 13).
                  For example, to mask
                  levels 0 - 13, use: /FFFC.
B    DC    /0000  levels to be masked. B
                  represents the second 10
                  levels (14 through 23).
                  For example, to mask
                  levels 14 through 23,
                  use : /FFCO.
```

Unmask:

```
CALL  UNMK
DC    A
DC    B
```

where A and B must be defined the same as shown for CALL MASK. The designated levels are unmasked.

Save Mask:

```
CALL  SAVMK
DC    A
DC    B
```

where A and B are the addresses of the core storage words where the contents of the interrupt mask register are to be placed.

Restore Mask:

```
CALL  RESMK
DC    A
DC    B
```

where A and B are the levels defined for the CALL MASK or CALL UNMK.

Reset Operation Monitor:

```
CALL  OPMON  no parameters are required
             for this call.
```

APPENDIX B. SUMMARY OF TSX STATEMENTS

Where Used Code	Statement	Description
I	CALL INTEX	Causes return of control to MIC on interrupt exit.
M	CALL CHAIN (NAME)	Mainline core load designated by NAME is loaded and executed.
M	CALL SPECL (NAME)	Mainline core load containing this call is saved on disk. Mainline core load designated by NAME is loaded and executed.
M	CALL BACK	Mainline core load saved as a result of a CALL SPECL is restored to core and execution continues with the statement following the special call.
M, I, N*, C	CALL QIFON (NAME, P, L, I, E)	Specified interrupts, that have been recorded, will be queued in the order called by the CALL QIFON statement and according to its designated parameters. NAME - name of the mainline core load. P - execution priority of the named mainline core load. L - interrupt level or indicator. I - PISW bit position indicator or CALL COUNT indicators. E - error parameter to specify the action to be taken if queue is full.
M, I, N*, C	CALL CLEAR (M, L, I, L, I, . . . , L, I,)	Specified interrupts will be cleared of recorded status whether they were recorded or not. M specifies the number of L and I parameters to follow. L and I are the same as designated for CALL QIFON. If M = 0, all recorded status is cleared.
M, I, N*, C	CALL QUEUE (NAME, P, E)	Mainline core load designated by NAME is entered in core load queue with priority P and error option E.
M, I, N*, C	CALL UNQ (NAME, P)	Mainline core load designated by priority P and NAME will be removed from the core load queue.

M	CALL VIAQ	Last logical statement of a mainline core load. The first core load, of the highest priority entered in the queue, is loaded and executed.
M, I, C	CALL MASK (I, J)	Interrupt levels specified by data statements for I and J are masked (no unmasking occurs).
M, I, C	CALL UNMK (I, J)	Interrupt levels specified by data statements for I and J are unmasked (no masking occurs).
M, I, N*, C	CALL SAVMK (I, J)	Interrupt level mask status is saved in I and J (no masking or unmasking occurs).
M, I, C	CALL RESMK (I, J)	Interrupt levels are masked according to I and J (all others are unmasked).
M, I, N, C	CALL OPMON	Operation monitor is reset.
M, I, N*, C	CALL TIMER (NAME, I, INT)	Interval timer specified by I (1 or 2) is set up to count INT intervals. After INT intervals have elapsed, ITC will branch to NAME (user's subprogram).
M, I, N*, C	CALL COUNT (IN, I, INB)	Program interval timer specified by I (1, 2, 3, . . . , 9) is set to count INB intervals. Upon completion of INB intervals, the ITC will branch to the subroutine specified by IN (IN specifies a subroutine number from 0 - 31).
M	CALL SHARE (I)	The present core load is saved and non-process time-sharing is set up for I timed intervals.
I, C	CALL ENDTS	Time sharing is terminated.
M, I, N*, C	CALL SETCL (I)	Programmed clock is set to equal I.
M, I, N*, C	CALL CLOCK (I)	Clock is read into I.
M, I, C	CALL LEVEL (L)	Calls the programmed interrupt specified by the hardware level L (L must be between 0 - 23).
M, I, C	CALL DPART	Tests the operation level of present use and, if an interrupt level exists, executes a CALL INTEX, otherwise a CALL VIAQ is executed.

M - Mainline programs only.
I - Interrupt programs only.
N - Nonprocess programs only.
C - Combination mainline and interrupt core load.
* - Must be an XEQ from core load area (TIME SHARING REQUIRED)
X - Must be an XEQ from core load area (TIME-SHARING NOT REQUIRED)

APPENDIX C. INTERRUPT LEVEL ASSIGNMENT CHART

INTERRUPT		PRIORITY LEVEL [1]	DECIMAL ADDRESS	ILSW	PISW [2] ASSIGN'T	MASK & UNMASK	PROGRAM INTERRUPT	I/O, TIMER, PROCESS INTERRUPT: ASSIGNMENT ALLOWED
BASIC	Internal	1	8	Yes	–	No	No	No
	Trace	26	9	No	–	[3]	No	No
	CE	27	1 [4]	No	–	No	No	No
	Assigned Levels 0	2	11	Yes	2	Yes	Yes	Yes
	1	3	12	Yes	3	Yes	Yes	Yes
	2	4	13	Yes	4	Yes	Yes	Yes
	3	5	14	Yes	5	Yes	Yes	Yes
	4	6	15	Yes	6	Yes	Yes	Yes
	5	7	16	Yes	7	Yes	Yes	Yes
	6	8	17	Yes	8	Yes	Yes	Yes
	7	9	18	Yes	9	Yes	Yes	Yes
	8	10	19	Yes	10	Yes	Yes	Yes
	9	11	20	Yes	11	Yes	Yes	Yes
	10	12	21	Yes	12	Yes	Yes	Yes
	11	13	22	Yes	13	Yes	Yes	Yes
SPECIAL FEATURE GROUP 1	12	14	23	Yes	14	Yes	Yes	Yes
	13	15	24	Yes	15	Yes	Yes	Yes
	14	16	25	Yes	16	Yes	Yes	Yes
	15	17	26	Yes	17	Yes	Yes	Yes
	16	18	27	Yes	18	Yes	Yes	Yes
	17	19	28	Yes	19	Yes	Yes	Yes
SPECIAL FEATURE GROUP 2	18	20	29	Yes	20	Yes	Yes	Yes
	19	21	30	Yes	21	Yes	Yes	Yes
	20	22	31	Yes	22	Yes	Yes	Yes
	21	23	32	Yes	23	Yes	Yes	Yes
	22	24	33	Yes	24	Yes	Yes	Yes
	23	25	34	Yes	25	Yes	Yes	Yes

[1] NOTE: 1 Highest priority
27 Lowest priority

[2] 24 PISW's Basic IBM 1800.

[3] Manually masked and unmasked by switch.

[4] Return address in I-counter stored in decimal address 0010, but hardware-generated BSI addresses decimal address 0001.

* Recognized by all Conversion subroutines
NOTE: Codes that are not asterisked are in error.

Ref No.	EBCDIC		IBM Card Code							Graphics and Control Names		1443 Printer Hex	PTTC/8 Hex U-Upper Case L-Lower Case	1053/1816 Printer Hex
	Binary	Hex	Rows						Hex					
	0123 4567		12	11	0	9	8	7-1						
0	0000 0000	00	12		0	9	8	1	B030	NUL				
1	0001	01	12			9		1	9010					
2	0010	02	12			9		2	8810					
3	0011	03	12			9		3	8410					
4	0100	04	12			9		4	8210	PF	Punch Off			
5*	0101	05	12			9		5	8110	HT	Horiz. Tab		6D ⑤	41 ①
6*	0110	06	12			9		6	8090	LC	Lower Case		6E ⑤	
7*	0111	07	12			9		7	8050	DEL	Delete		7F ⑤	
8	1000	08	12			9	8		8030					
9	1001	09	12			9	8	1	9030					
10	1010	0A	12			9	8	2	8830					
11	1011	0B	12			9	8	3	8430					
12	1100	0C	12			9	8	4	8230					
13	1101	0D	12			9	8	5	8130					
14	1110	0E	12			9	8	6	8080					
15	1111	0F	12			9	8	7	8070					
16	0001 0000	10	12	11		9	8	1	D030					
17	0001	11		11		9		1	5010					
18	0010	12		11		9		2	4810					
19	0011	13		11		9		3	4410					
20*	0100	14		11		9		4	4210	RES	Restore		4C ⑤	05 ②
21*	0101	15		11		9		5	4110	NL	New Line		DD ⑤	81 ③
22*	0110	16		11		9		6	4090	BS	Backspace		5E ⑤	11
23	0111	17		11		9		7	4050	IDL	Idle			
24	1000	18		11		9	8		4030					
25	1001	19		11		9	8	1	5030					
26	1010	1A		11		9	8	2	4830					
27	1011	1B		11		9	8	3	4430					
28	1100	1C		11		9	8	4	4230					
29	1101	1D		11		9	8	5	4130					
30	1110	1E		11		9	8	6	40B0					
31	1111	1F		11		9	8	7	4070					
32	0010 0000	20		11	0	9	8	1	7030					
33	0001	21			0	9		1	3010					
34	0010	22			0	9		2	2810					
35	0011	23			0	9		3	2410					
36	0100	24			0	9		4	2210	BYP	Bypass			
37*	0101	25			0	9		5	2110	LF	Line Feed		3D ⑤	03
38*	0110	26			0	9		6	2090	EOB	End of Block		3E ⑤	
39	0111	27			0	9		7	2050	PRE	Prefix			
40	1000	28			0	9	8		2030					
41	1001	29			0	9	8	1	3030					
42	1010	2A			0	9	8	2	2830					
43	1011	2B			0	9	8	3	2430					
44	1100	2C			0	9	8	4	2230					
45	1101	2D			0	9	8	5	2130					
46	1110	2E			0	9	8	6	20B0					
47	1111	2F			0	9	8	7	2070					
48	0011 0000	30	12	11	0	9	8	1	F030					
49	0001	31				9		1	1010					
50	0010	32				9		2	0810					
51	0011	33				9		3	0410					
52	0100	34				9		4	0210	PN	Punch On			
53*	0101	35				9		5	0110	RS	Reader Stop		0D ⑤	09 ④
54*	0110	36				9		6	0090	UC	Upper Case		0E ⑤	
55	0111	37				9		7	0050	EOT	End of Trans.			
56	1000	38				9	8		0030					
57	1001	39				9	8	1	1030					
58	1010	3A				9	8	2	0830					
59	1011	3B				9	8	3	0430					
60	1100	3C				9	8	4	0230					
61	1101	3D				9	8	5	0130					
62	1110	3E				9	8	6	00B0					
63	1111	3F				9	8	7	0070					

NOTES: Typerwriter Output

① Tabulate
② Shift to black
③ Carrier Return
④ Shift to red
⑤ The same in either upper or lower case.

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	IBM Card Code Hex	Graphics and Control Names	1443 Printer Hex	PTTC/8 Hex U-Upper Case L-Lower Case	1053/1816 Printer Hex
64*	0100	0000	40	no punches						0000	(space)	00	10 ⑤	21
65		0001	41	12		0	9		1	B010				
66		0010	42	12		0	9		2	A810				
67		0011	43	12		0	9		3	A410				
68		0100	44	12		0	9		4	A210				
69		0101	45	12		0	9		5	A110				
70		0110	46	12		0	9		6	A090				
71		0111	47	12		0	9		7	A050				
72		1000	48	12		0	9	8		A030				
73		1001	49	12				8	1	9020				
74*		1010	4A	12				8	2	8820	¢	3D	20 (U)	02
75*		1011	4B	12				8	3	8420	. (period)	3B	6B (L)	00
76*		1100	4C	12				8	4	8220	<	3E	02 (U)	DE
77*		1101	4D	12				8	5	8120	(	1C	19 (U)	FE
78*		1110	4E	12				8	6	80A0	+	10	70 (U)	DA
79*		1111	4F	12				8	7	8060	| (logical OR)	3F	3B (U)	C6
80*	0101	0000	50	12						8000	&	30	70 (L)	44
81		0001	51	12	11		9		1	D010				
82		0010	52	12	11		9		2	C810				
83		0011	53	12	11		9		3	C410				
84		0100	54	12	11		9		4	C210				
85		0101	55	12	11		9		5	C110				
86		0110	56	12	11		9		6	C090				
87		0111	57	12	11		9		7	C050				
88		1000	58	12	11		9	8		C030				
89		1001	59		11			8	1	5020				
90*		1010	5A		11			8	2	4820	!	2D	5B (U)	42
91*		1011	5B		11			8	3	4420	$	2B	5B (L)	40
92*		1100	5C		11			8	4	4220	*	2C	08 (U)	D6
93*		1101	5D		11			8	5	4120	)	3C	1A (U)	F6
94*		1110	5E		11			8	6	40A0	;	2E	13 (U)	D2
95*		1111	5F		11			8	7	4060	¬ (logical NOT)	2F	6B (U)	F2
96*	0110	0000	60		11					4000	- (dash)	20	40 (L)	84
97*		0001	61			0			1	3000	/	11	31 (L)	BC
98		0010	62		11	0	9		2	6810				
99		0011	63		11	0	9		3	6410				
100		0100	64		11	0	9		4	6210				
101		0101	65		11	0	9		5	6110				
102		0110	66		11	0	9		6	6090				
103		0111	67		11	0	9		7	6050				
104		1000	68		11	0	9	8		6030				
105		1001	69			0		8	1	3020				
106		1010	6A	12	11					C000				
107*		1011	6B			0		8	3	2420	, (comma)	1B	3B (L)	80
108*		1100	6C			0		8	4	2220	%	1A	15 (U)	06
109*		1101	6D			0		8	5	2120	_ (underscore)	1D	40 (U)	BE
110*		1110	6E			0		8	6	20A0	>	0E	07 (U)	46
111*		1111	6F			0		8	7	2060	?	0F	31 (U)	86
112	0111	0000	70	12	11	0				E000				
113		0001	71	12	11	0	9		1	F010				
114		0010	72	12	11	0	9		2	E810				
115		0011	73	12	11	0	9		3	E410				
116		0100	74	12	11	0	9		4	E210				
117		0101	75	12	11	0	9		5	E110				
118		0110	76	12	11	0	9		6	E090				
119		0111	77	12	11	0	9		7	E050				
120		1000	78	12	11	0	9	8		E030				
121		1001	79					8	1	1020				
122*		1010	7A					8	2	0820	:	1E	04 (U)	82
123*		1011	7B					8	3	0420	#	2A	0B (L)	C0
124*		1100	7C					8	4	0220	@	0C	20 (L)	04
125*		1101	7D					8	5	0120	' (apostrophe)	0D	16 (U)	E6
126*		1110	7E					8	6	00A0	=	0B	01 (U)	C2
127*		1111	7F					8	7	0060	"	1F	0B (U)	E2

Ref No.	EBCDIC Binary	EBCDIC Hex	IBM Card Code Rows						IBM Card Code Hex	Graphics and Control Names	1443 Printer Hex	PTTC/8 Hex U-Upper Case L-Lower Case	1053/1816 Printer Hex
	0123 4567		12	11	0	9	8	7-1					
128	1000 0000	80	12		0		8	1	B020				
129	0001	81	12		0			1	B000	a			
130	0010	82	12		0			2	A800	b			
131	0011	83	12		0			3	A400	c			
132	0100	84	12		0			4	A200	d			
133	0101	85	12		0			5	A100	e			
134	0110	86	12		0			6	A080	f			
135	0111	87	12		0			7	A040	g			
136	1000	88	12		0		8		A020	h			
137	1001	89	12		0	9			A010	i			
138	1010	8A	12		0		8	2	A820				
139	1011	8B	12		0		8	3	A420				
140	1100	8C	12		0		8	4	A220				
141	1101	8D	12		0		8	5	A120				
142	1110	8E	12		0		8	6	A0A0				
143	1111	8F	12		0		8	7	A060				
144	1001 0000	90	12	11			8	1	D020				
145	0001	91	12	11				1	D000	j			
146	0010	92	12	11				2	C800	k			
147	0011	93	12	11				3	C400	l			
148	0100	94	12	11				4	C200	m			
149	0101	95	12	11				5	C100	n			
150	0110	96	12	11				6	C080	o			
151	0111	97	12	11				7	C040	p			
152	1000	98	12	11			8		C020	q			
153	1001	99	12	11		9			C010	r			
154	1010	9A	12	11			8	2	C820				
155	1011	9B	12	11			8	3	C420				
156	1100	9C	12	11			8	4	C220				
157	1101	9D	12	11			8	5	C120				
158	1110	9E	12	11			8	6	C0A0				
159	1111	9F	12	11			8	7	C060				
160	1010 0000	A0		11	0		8	1	7020				
161	0001	A1		11	0			1	7000				
162	0010	A2		11	0			2	6800	s			
163	0011	A3		11	0			3	6400	t			
164	0100	A4		11	0			4	6200	u			
165	0101	A5		11	0			5	6100	v			
166	0110	A6		11	0			6	6080	w			
167	0111	A7		11	0			7	6040	x			
168	1000	A8		11	0		8		6020	y			
169	1001	A9		11	0	9			6010	z			
170	1010	AA		11	0		8	2	6820				
171	1011	AB		11	0		8	3	6420				
172	1100	AC		11	0		8	4	6220				
173	1101	AD		11	0		8	5	6120				
174	1110	AE		11	0		8	6	60A0				
175	1111	AF		11	0		8	7	6060				
176	1011 0000	B0	12	11	0		8	1	F020				
177	0001	B1	12	11	0			1	F000				
178	0010	B2	12	11	0			2	E800				
179	0011	B3	12	11	0			3	E400				
180	0100	B4	12	11	0			4	E200				
181	0101	B5	12	11	0			5	E100				
182	0110	B6	12	11	0			6	E080				
183	0111	B7	12	11	0			7	E040				
184	1000	B8	12	11	0		8		E020				
185	1001	B9	12	11	0	9			E010				
186	1010	BA	12	11	0		8	2	E820				
187	1011	BB	12	11	0		8	3	E420				
188	1100	BC	12	11	0		8	4	E220				
189	1101	BD	12	11	0		8	5	E120				
190	1110	BE	12	11	0		8	6	E0A0				
191	1111	BF	12	11	0		8	7	E060				

Ref No.	EBCDIC			IBM Card Code							Graphics and Control Names	1443 Printer Hex	PTTC/8 Hex U-Upper Case L-Lower Case	1053/1816 Printer Hex
	Binary		Hex	Rows						Hex				
	0123	4567		12	11	0	9	8	7-1					
192	1100	0000	C0	12		0				A000	(+ zero)			
193*		0001	C1	12					1	9000	A	31	61 (U)	~~3C or~~ 3E
194*		0010	C2	12					2	8800	B	32	62 (U)	~~18 or~~ 1A
195*		0011	C3	12					3	8400	C	33	73 (U)	~~1C or~~ 1E
196*		0100	C4	12					4	8200	D	34	64 (U)	~~30 or~~ 32
197*		0101	C5	12					5	8100	E	35	75 (U)	~~34 or~~ 36
198*		0110	C6	12					6	8080	F	36	76 (U)	~~10 or~~ 12
199*		0111	C7	12					7	8040	G	37	67 (U)	~~14 or~~ 16
200*		1000	C8	12				8		8020	H	38	68 (U)	~~24 or~~ 26
201*		1001	C9	12			9			8010	I	39	79 (U)	~~20 or~~ 22
202		1010	CA	12		0	9	8	2	A830				
203		1011	CB	12		0	9	8	3	A430				
204		1100	CC	12		0	9	8	4	A230				
205		1101	CD	12		0	9	8	5	A130				
206		1110	CE	12		0	9	8	6	A0B0				
207		1111	CF	12		0	9	8	7	A070				
208	1101	0000	D0		11	0				6000	(- zero)			
209*		0001	D1		11				1	5000	J	21	51 (U)	~~7C or~~ 7E
210*		0010	D2		11				2	4800	K	22	52 (U)	~~58 or~~ 5A
211*		0011	D3		11				3	4400	L	23	43 (U)	~~5C or~~ 5E
212*		0100	D4		11				4	4200	M	24	54 (U)	~~70 or~~ 72
213*		0101	D5		11				5	4100	N	25	45 (U)	~~74 or~~ 76
214*		0110	D6		11				6	4080	O	26	46 (U)	~~50 or~~ 52
215*		0111	D7		11				7	4040	P	27	57 (U)	~~54 or~~ 56
216*		1000	D8		11			8		4020	Q	28	58 (U)	~~64 or~~ 66
217*		1001	D9		11		9			4010	R	29	49 (U)	~~60 or~~ 62
218		1010	DA	12	11		9	8	2	C830				
219		1011	DB	12	11		9	8	3	C430				
220		1100	DC	12	11		9	8	4	C230				
221		1101	DD	12	11		9	8	5	C130				
222		1110	DE	12	11		9	8	6	C0B0				
223		1111	DF	12	11		9	8	7	C070				
224	1110	0000	E0			0		8	2	2820				
225		0001	E1		11	0	9		1	7010				
226*		0010	E2			0			2	2800	S	12	32 (U)	~~98 or~~ 9A
227*		0011	E3			0			3	2400	T	13	23 (U)	~~9C or~~ 9E
228*		0100	E4			0			4	2200	U	14	34 (U)	~~B0 or~~ B2
229*		0101	E5			0			5	2100	V	15	25 (U)	~~B4 or~~ B6
230*		0110	E6			0			6	2080	W	16	26 (U)	~~90 or~~ 92
231*		0111	E7			0			7	2040	X	17	27 (U)	~~94 or~~ 96
232*		1000	E8			0		8		2020	Y	18	38 (U)	~~A4 or~~ A6
233*		1001	E9			0	9			2010	Z	19	29 (U)	~~A0 or~~ A2
234		1010	EA		11	0	9	8	2	6830				
235		1011	EB		11	0	9	8	3	6430				
236		1100	EC		11	0	9	8	4	6230				
237		1101	ED		11	0	9	8	5	6130				
238		1110	EE		11	0	9	8	6	60B0				
239		1111	EF		11	0	9	8	7	6070				
240*	1111	0000	F0			0				2000	0	0A	1A (L)	C4
241*		0001	F1						1	1000	1	01	01 (L)	FC
242*		0010	F2						2	0800	2	02	02 (L)	D8
243*		0011	F3						3	0400	3	03	13 (L)	DC
244*		0100	F4						4	0200	4	04	04 (L)	F0
245*		0101	F5						5	0100	5	05	15 (L)	F4
246*		0110	F6						6	0080	6	06	16 (L)	D0
247*		0111	F7						7	0040	7	07	07 (L)	D4
248*		1000	F8					8		0020	8	08	08 (L)	E4
249*		1001	F9				9			0010	9	09	19 (L)	E0
250		1010	FA	12	11	0	9	8	2	E830				
251		1011	FB	12	11	0	9	8	3	E430				
252		1100	FC	12	11	0	9	8	4	E230				
253		1101	FD	12	11	0	9	8	5	E130				
254		1110	FE	12	11	0	9	8	6	E0B0				
255		1111	FF	12	11	0	9	8	7	E070				

APPENDIX E. SUBROUTINE LIBRARY

Subroutine	Names	Other Subroutines Required
FORTRAN		
Called by CALL		
Sense and Program Select Switches	SSWTC	None
Test Data Entry Switches	DATSW	None
Sense Light Control and Test	SLITE, SLITT	None
Overflow Test	OVERF	None
Divide Check Test	DVCHK	None
Functional Error Test	FCTST	None
Trace Start	TSTRT	TSET
Trace Stop	TSTOP	TSET
Integer Transfer of Sign	ISIGN	None
Transfer of Sign (Extended Precision)	ESIGN	ESUB, ELD
Transfer of Sign (Standard Precision)	FSIGN	FSUB, FLD
Analog Input-Single Read (point)	AIP	AIPTN, GAGED, QZ010, QZERQ
Analog Input-Sequential Read (table)	AIS	AISQN, GAGED, QZ010, QZERQ
Analog Input-Random Read (table)	AIR	AIRNN, GAGED, QZ010, QZERQ
Contact Operate, Digital Output, Digital-Analog Conversion, and Pulse Output	CO, DO, DAC, PO	DAOP, GAGED, QZ010, QZERQ
Contact Sense, Voltage Level Sense, Pulse Output, and Digital Input	CS, VS, PI, DI	DINP, GAGED, QZ010, QZERQ
Contact Sense and Compare, Voltage Level Sense and Compare, Pulse Input and Compare, and Digital Input and Compare	CSC, VSC, PIC, DIC	DICMP, XSAVE, IOPE, QZERQ
Contact Sense and Expand, Voltage Level Sense and Expand, Pulse Input and Expand, and Digital Input and Expand	CSX, VSX, PIX, DIX	DIEXP, GAGED, QZ010, QZERQ
Save, Restore, and Error Handling	QZERQ, QZ010, XSAVE, XEXIT, XLOAD, ETS, IOPE, OULSY, UNGAG, GAGED	None
Called by LIBF		
Magnetic Tape Control (multiple device)	REWND, BCKSP, EOF	IOU, MAGT
Address Check	ADRCK	None
Computed GOTO Check	COMGG, COMGI	None
Indexed Integer Store	ISTOX	ADRCK
Transfer Trace	MGOTO, MFIF, MIIF, MEIF	TTEST, COMGI, MWRT, MIOI, MIOF, MCOMP
Trace Test and Set	TTEST, TSET	None
Pause	PAUSE	None
Stop	STOP	EXIT
Subscript Displacement Calculation	SUBSC	None
Subroutine Initialization	SUBIN	None
Logical to Physical Unit Cross Reference Table	IOU	None
Interface Routines for FAC	LDFAC, STFAC, SBFAC, DVFAC	None
Disk FORTRAN I/O	MDFIO, MDAF, MDAI, MDCOM, MDF, MDI, MDFX, MDIX, MDRED, MDWRT	ADRCK, DISKN, BT2BT, SAVE, IOFIX
Disk FIND	MDFND	DISKN

Subroutine	Names	Other Subroutines Required
FORTRAN (continued)		
Called by LIBF (continued)		
Non-Disk FORTRAN I/O	MFIO, MRED, MWRT, MCOMP, MIOAF, MIOAI, MIOF, MIOI, MIOFX, MIOIX	DISKN, IOU, IOFIX, BT1BT, SAVE, ADRCK, FLOAT, IFIX
Buffer Save and Get Precision and Integer Size	SAVE, IOFIX	None
Arithmetic Trace	MIAR, MIARX, MFAR, MFARX, MEAR, MEARX	ADRCK, TTEST, MWRT, MIOI, MIOF, MCOMP
ARITHMETIC AND FUNCTIONAL*		
Called by CALL		
Real Hyperbolic Tangent (E)	ETANH, ETNH	EEXP, ELD/ESTO, EADD, EDIV
Real Hyperbolic Tangent (S)	FTANH, FTNH	FEXP, FLD/FSTO, FADD, FDIV
Real Base to a Real Exponent (E)	EAXB, EAXBX	EEXP, ELN, EMPY
Real Base to a Real Exponent (S)	FAXB, FAXBX	FEXP, FLN, EMPY
Real Natural Logarithm (E)	ELN, EALOG	XMD, EADD, EMPY, EDIV, NORM
Real Natural Logarithm (S)	FLN, FALOG	FSTO, XMDS, FADD, FMPY, FDIV, NORM
Real Exponential (E)	EEXP, EXPN	XMD, FARC
Real Exponential (S)	FEXP, FXPN	XMDS, FARC
Real Square Root (E)	ESQR, ESQRT	ELD/ESTO, EADD, EMPY, EDIV
Real Square Root (S)	FSQR, FSQRT	FLD/FSTO, FADD, FMPY, FDIV
Real Trigonomentric Sine/Cosine (E)	ESIN/ECOS, ESINE/ECOSN	EADD, EMPY, NORM, XMD, ESTO
Real Trigonometric Sine/Cosine (S)	FSIN/FCOS, FSINE/FCOSN	FADD, FMPY, NORM, XMDS, FSTO
Real Trigonometric Arctangent (E)	EATN, EATAN	EADD, EMPY, EDIV, XMD, NORM
Real Trigonometric Arctangent (S)	FATN, FATAN	FADD, FMPY, FDIV, XMDS, FSTO
Fixed-Point Square Root	XSQR	None
Real Absolute Value (E)	EABS, EAVL	None
Real Absolute Value (S)	FABS, FAVL	None
Integer Absolute Value	IABS	None
Real Binary to Decimal/ Real Decimal to Binary	FBTD/FDTB	None
Save and Restore (E)	ETRTN, ETNTR	None
Save and Restore (S)	FTRTN, FTNTR	None
AND	IAND	None
OR	IOR	None
EOR	IEOR	None
LD	LD	None
Called by LIBF		
Real Base to an Integer Exponent (E)	EAXI, EAXIX	ELD/ESTO, EMPY, EDVR, ETNTR, ETRTN
Real Base to an Integer Exponent (S)	FAXI, FAXIX	FLD/FSTO, FMPY, FDVR, FTNTR, FTRTN
Real Divide (E)	EDIV, EDIVX, EDVR, EDVRX	XDD, FARC, ETNTR, ETRTN, FLD/FSTO
Real Divide (S)	FDIV, FDIVX, FDVR, FDVRX	FARC, ELD/ESTO, FTRTN, FTNTR
Real Multiply (E)	EMPY, EMPYX	XMD, FARC, ETNTR, ETRTN
Real Multiply (S)	FMPY, FMPYX	XMDS, FARC, FTNTR, FTRTN
Real Add/Subtract and Reverse Subtract (E)	EADD, EADDX/ESUB, ESUBX, FSBR, ESBRX	FARC, NORM, ETNTR, ETRTN

*The word "real" is synonymous with the word floating point.

Subroutine	Names	Other Subroutines Required
ARITHMETIC AND FUNCTIONAL (continued)		
Called by LIBF (continued)		
Real Add/Subtract and Reverse Subtract (S)	FADD, FADDX/FSUB, FSUBX FSBR, FSBRX	FARC, NORM, FTNTR, FTRTN
Load/Store FAC (E)	ELD, ELDX/ESTO, ESTOX	ADRCK
Load/Store FAC (S)	FLD, FLDX/FSTO, FSTOX	ADRCK
Fixed-Point Double-Word Divide	XDD	XMD
Fixed-Point Double-Word Multiply	XMD	None
Fixed-Point Fractional Multiply	XMDS	None
Real Reverse Sign	SNR	None
Integer to Real	FLOAT	NORM
Real to Integer	IFIX	None
Integer Base to an Integer Exponent	FIXI, FIXIX	None
Normalize	NORM	None
Real Arithmetic Range Check	FARC	None
INTERRUPT SERVICE		
Called by LIBF		
Card I/O	CARDN	None
Disk I/O	DISKN	None
Paper Tape I/O	PAPTN	None
Plotter Control and Output	PLOTX	None
1443 Printer Output, Overlapping	PRNTN	None
1816/1053 Printer-Keyboard I/O, Overlapping (Error Parameter)	TYPEN	None
1816/1053 Printer Output, Overlapping (Error Parameter)	WRTYN	None
Magnetic Tape I/O	MAGT	None
Digital Input	DINP	None
Digital Input Read/Compare	DICMP	IOPE
Digital Input Read/Expand	DIEXP	GAGED
Digital/Analog Output	DAOP	GAGED
Analog Input Interrupt Processing	ANINT	None
Single Point Analog Input	AIPTN, AIPN	ANINT, GAGED
Sequential Analog Input	AISQN, AISN	ANINT, GAGED
Random Analog Input	AIRN	ANINT, GAGED
I/O Busy Test	BT1BT	None
I/O Busy Test	BT2BT	None
CONVERSION		
Called by LIBF		
Binary Word to 6 Decimal Characters (card code)	BINDC	None
Binary Word to 4 Hexadecimal Characters (card code)	BINHX	None
6 Decimal Characters (card code) to Binary Word	DCBIN	None
EBCDIC to 1053 or 1443 Printer Output Code	EBPRT	PRT
IBM Card Code to or from EBCDIC	HOLEB	None
IBM Card Code to 1053 or 1443 Printer Output Code	HOLPR	PRT

Subroutine	Names	Other Subroutines Required
CONVERSION (continued)		
Called by LIBF (continued)		
4 Hexadecimal Characters (card code) to Binary Word	HXBIN	None
EBCDIC to or from PTTC/8	PAPEB	EBPA
IBM Card Code to or from PTTC/8	PAPHL	EBPA
PTTC/8 to 1053 or 1443 Printer Code	PAPPR	EBPA
EBCDIC and PTTC/8 Table	EBPA	None
1053 or 1443 Printer Code Table	PRT	None
MISCELLANEOUS SUBROUTINES		
Called by Call		
Clear Recorded Interrupts	CLEAR	None
Read the Time Clock	CLOCK	None
Programmed Timers	COUNT	None
Exit from Combination Core Loads	DPART	VIAQ INTEX
End Time Sharing	ENDTS	None
Programmed Interrupt	LEVEL	None
Mask Interrupt Levels	MASK	None
Reset Operations Monitor	OPMON	None
Service Recorded Interrupts	QIFON	QUEUE
Queue Mainline Core Loads	QUEUE	None
Restore Mask	RESMK	None
Save Mask	SAVMK	None
Set the Time Clock	SETCL	None
Hardware Timers	TIMER	None
Unmask Interrupt Levels	UNMK	None
Remove Mainline from Queue	UNQ	None
Load Highest Priority Mainline	VIAQ	CHAIN, SHARE
Dump Status	DMPS, DMPST	None
Selective Dump	DMP, DMPHX, DMPDC	None
Trace Hex Conversion	CONHX	None
Trace Print	TRPRT	None

APPENDIX F. SUBROUTINE CORE REQUIREMENTS

Tables 9 through 14 list the core requirements for the functional, FORTRAN, I/O, conversion, plotter, and miscellaneous subroutines, respectively.

Table 9. Arithmetic and Functional Subroutine Core Requirements

Standard Precision		Extended Precision	
Subroutines	No of. Core Locations	Subroutines	No. of Core Locations
FADD, FADDX, FSBR, FSBRX, FSUB, FSUBX	158	EADD, EADDX, ESBR, ESBRX, ESUB, ESUBX	158
FALOG, FLN	148	EALOG, ELN	164
FATAN, FATN	170	EATAN, EATN	184
FAVL, FABS	28	EAVL, EABS	28
FAXB, FAXBX	70	EAXB, EAXBX	74
FAXI, FAXIX	68	EAXI, EAXIX	72
FDIV, FDIVX, FDVR, FDVRX	106	EDIV, EDIVX, EDVR, EDVRX	98
FEXP, FXPN	124	EEXP, EXPN	154
FLD, FLDX, FSTO, FSTOX	102	ELD, ELDX, ESTO, ESTOX	107
FMPY, FMPYX	65	EMPY, EMPYX	42
FSIN, FSINE, FCOS, FCOSN	148	ESIN, ESINE, ECOS, ECOSN	180
FSQRT, FSQR	86	ESQRT, ESQR	90
FTANH, FTNH	84	ETANH, ETNH	76
FTRTN, FTNTR	40	ETRTN, ETNTR	44
Either Precision			
Subroutines	No. of Core Locations	Subroutines	No. of Core Locations
FARC	52	IOR	14
FBTD, FDTB	428	LD	16
FLOAT	26	NORM	44
FIXI, FIXIX	68	SNR	24
IABS	26	XDD	78
IAND	14	XMD	58
IEOR	14	XMDS	42
IFIX	46	XSQR	50

Table 10. FORTRAN Subroutine Core Requirements

Subroutines	No. of Core Locations
ADRCK	82
COMGG, COMGI	82
DATSW	44
DVCHK	20
ESIGN (EXTENDED PRECISION)	60
FSIGN (STANDARD PRECISION)	60
FCTST	38
IOU	90
ISIGN	34
ISTOX	32
LDFAC, STFAC, SBFAC, DVFAC	46
MDFIO, MDAF, MDAI, MDCOM, MDF, MDFX, MDI, MDIX, MDRED, MDWRT	883
MDFND	120
MFIO, MRED, MWRT, MCOMP, MIOAF, MIOIX, MIOAI, MIOI, MIOFX, MIOF	1595
MGOTO, MFIF, MIIF, MEIF	208
MIAR, MIARX, MFAR, MFARX, MEAR, MEARX	200
OVERF	20
PAUSE	16
SAVE, IOFIX	116
SLITE, SLITT	76
SSWTC	42
STOP	26
SUBIN	36
SUBSC	44
STOP	26
TSTOP	14
TSTRT	14
TTEST, TSET	26
UFIO	577

Table 11. I/O Subroutine Core Requirements

Subroutines	No. of Core Locations
CARDN	356
PAPTN	258
MAGT	542
PLOTX	222
REWIND/BCKSP/EOF	62
DAOP	298
AIPTN, AIPN	130
AISQN, AISN	246
AIRN	212
DIEXP	76
DICMP	92
DINP	298
ANINT	316
AIP	44
AIS	204
AIR	276
CO, DO, PO, DAC	92
CS, VS, PI, DI	100
CSC, VSC, PIC, DIC	146
CSX, VSX, PIX, DIX	38
IOPE, OUSLY, ETS	206
XSAVE, XEXIT, XLOAD	270
GAGED, UNGAG	32
QZERQ	14
QZ010	84
BT1BT	80
BT2BT	28

● Table 12. Conversion Subroutine Core Requirements

Subroutines	No. of Core Locations
BINDC	78
DCBIN	88
BINHX	52
HXBIN	70
HOLEB	290
HOLPR	196
EBPRT	160
PAPEB	264
PAPHL	330
PAPPR	272
EBPA	76
PRT	74

Table 13. Plotter Subroutine Core Requirements

Subroutines	No. of Core Locations
FCHAR	52
SCALF	4
FGRID	102
FPLOT	48
ECHAR	58
SCALE	4
EGRID	110
EPLOT	56
POINT	108
FCHRX, FCHRI, WCHRI,	596
FRULE, FMOVE, FINC	130
ECHRX, ECHRI, VCHRI	604
ERULE, EMOVE, EINC	144
XYPLT	100
PLOTI, PLOTS	28

● Table 14. Miscellaneous Subroutine Core Requirements

Subroutines	No. of Core Locations
DMPHX, DMP, DMPDC	426
DMPS, DMPST	318
DPART	14
ENDTS	10
LEVEL	56
MASK	38
OPMON	6
QIFON	154
QUEUE	194
RESMK	44
SAVMK	26
SETCL	26
TIMER	86
UNMK	74
UNQ	70
VIAQ	96
COUNT	52
CLOCK	18
CLEAR	128
CONHX	78
TRPRT	86
FLIP	59

INDEX

IBM Technical Newsletter

File Number	1800-36
Re: Form No.	C26-5990-1
This Newsletter No.	N26-0560
Date	October 25, 1966
Previous Newsletter Nos.	N26-0559

IBM 1800 Time-Sharing Executive System Specifications (Form C26-5990-1)

The attached pages and listed changes to existing pages bring the above publication up to date. Changes on replacement pages are indicated by a vertical line at the left of affected text, a bullet (●) at the left of the title of a changed illustration, and a bullet beside the page number of a page that should be reviewed in its entirety. Pages that contain changes are coded in the upper outside corner.

Note the following corrections on page 3 of 4, TNL N26-0559.

Page 104, Col. 2	Page reference should have been 103

Replace the following pages:

Cover and ii	109 and 110
iii and blank	116. 1 and blank
23 and 24	119 through 120. 2
25 and 26	120. 3 and blank
33 and 34	127 and blank
37 and 38	132. 5 and 132. 6
51 through 56	
59 through 70	
75 and 76	
89 and 90	
91 and 92	

Add the following pages:

38. 1 and blank

File this Newsletter at the back of the manual. It will provide a reference to changes, a method of determining that all amendments have been received, and a check for determining if the manual contains the proper pages.

IBM Corporation, Programming Publications, Dept. 452, San Jose, Calif. 95114

Make the following changes on the pages indicated:

Page 5, Col. 2

Under the heading "LOCAL Subprograms", change the first sentence to read as follows:
LOCAL subprograms are read into core storage by an overlay routine (FLIP) for execution when called by the object program.
Add the following paragraph at the bottom of the column.
Skeleton TV. The Executive Transfer Vector (ETV) serves as a link between in-skeleton routines and in-skeleton programs.

Page 7, Col. 2, Line 8

Delete the following line:
4. Trace and C. E. level.

Page 11, Col. 1

Add the following sentence after the word "interrogates" in the last line of the column:
The QIFON subroutine then automatically clears the interrogated program indicator.

Page 18, Col. 1, Line 12

Change to read as follows:
I/O device interrupt level.

Col. 2

After the statement "Any combination can be used for PISW assignments on levels 4 and 5," insert the following paragraph:
The user written routine used to service the interrupts must be coded as an I/O RPQ subroutine.

Page 27, Table 2

In the DEVICE/ROUTINE column change the DISK I/O and NON-DISK I/O entries as follows:
DISK I/O (FORTRAN)
NON-DISK I/O (FORTRAN)

Page 29, Col. 2

In the second paragraph under Simulator Program, change the second sentence to read as follows:
For example, an analog input call sequence can obtain input from either cards or a random number generator.

Page 30, Col. 1

Change the first three lines of the last paragraph to read as follows:
When nonprocess jobs are ready to be executed, the operator places the input in the card reader, sets the required configuration of the

Page 32, Col. 2

Change the heading *STOREMOD to *STOREMD

Page 44, Col. 1

Change the first line of the note at the bottom of the column to read as follows:
NOTE: The following require a data channel per device:

Page 45, Col. 1

Under Special Condition Parameter add the following paragraph:

If the I/O area and the I/O subroutine are in the system skeleton, the special condition routine must also be in the skeleton.

Page 48, Col. 2 — Add the following statement to the note at the top of the column <u>and</u> to the paragraph headed <u>Write Immediate</u>:
Writing a partial sector clears the remainder of the sector to zero.

Page 49, Col. 1 — Under <u>I/O Area Parameter,</u> change line 6 to read as follows:
sector or cylinder size, since DISKN crosses

Page 73, Col. 2 — Under "I/O Function" change line 2 to read as follows:
performed by the AIPTN subroutines. The functions,

Page 82, Col. 1, Line 4 — Change to read as follows:
IM (403), IK (403), IF007 (2)

Last line — NTABL (202), IM(1), IM(403), IK(1), IK(403), IF007(2), LIM)

Page 83, Col. 2 — Under DATA TABLE change the last two margin entries to read as follows:
IF007(2)
IF007(1)

Page 87, Col. 1 — Change the first paragraph to read as follows:
This addressing mode digit specifies one of four available addressing options and is valid only for data channel operations. Program channel is always in sequential mode.

Page 88, Col. 2 — Under NOTE at the bottom of the column, change the second sentence to read as follows:
The only difference is the appearance of the data table in which only one variable position would be designated as an address point.

Page 103, Col. 1 — Change the first illustration as follows:

1st word	S	15 most significant bits of mantissa
	0	1 15

Page 104, Table 7 — In the <u>Extended Precision</u> column change ETAHN to ETANH.

Page 105, Col. 1 — Change the text under the illustration as follows:
FAC
(XR3+41)

Page 114, Col. 2 — Change the heading FLOATING-POINT BASE TO AN INTEGER EXPONENT to read as follows:
REAL BASE TO A REAL EXPONENT

Page 132.1, Appendix E — Under the column <u>Names</u>, change COMGG to COMGO.

IBM Technical Newsletter

File Number 1800-36

Re: Form No. C26-5990-1

This Newsletter No. N26-0523

Date January 11, 1967

Previous Newsletter Nos. N26-0558
N26-0559

IBM 1800 Time-Sharing Executive System Specifications (Form C26-5990-1)

Two restrictions that do not apply to TSX version two have been removed from the attached page. Changes are indicated by a vertical line at the left of the affected text.

Replace the following pages:

17 and 18

File this Newsletter at the back of the manual. It will provide a reference to changes, a method of determining that all amendments have been received, and a check for determining if the manual contains the proper pages.

IBM Corporation, Programming Publications, Dept. 232, San Jose, Calif. 95114

PRINTED IN U.S.A.

IBM Technical Newsletter

File Number 1800-36

Re: Form No. C26-5990-1

This Newsletter No. N26-0559

Date July 20, 1966

Previous Newsletter Nos. N26-0558

IBM 1800 Time-Sharing Executive System Specifications (Form C26-5990-1)

The attached pages and the listed changes to existing pages bring the above publication up to date. Changes on replacement pages are indicated by a vertical line at the left of affected text, a bullet (o) at the left of the title of a changed illustration, and a bullet beside the page number of a page that should be reviewed in its entirety. Pages that contain changes are coded in the upper outside corner.

Replace the following pages:

Cover and ii	61 and 62
3 and 4	63 and 64
5 and 6	65 and 66
7 and 8	67 and 68
13 and 14	69 and 70
15 and 16	71
17 and 18	75 and 76
23 and 24	79 and 80
25 and 26	81 and 82
27 and 28	85 and 86
31 and 32	87 and 88
33 and 34	89 and 90
35 and 36	91 and 92
37 and 38	95 and 96
39 and 40	115 and 116
43 and 44	117 and 118
45 and 46	119 and 120
49 and 50	125 and 126
55 and 56	

Add the following pages:

116.1	132.1 and 132.2
120.1 and 120.2	132.3 and 132.4
120.3	132.5 and 132.6

Make the following changes on the pages indicated:

IBM Corporation, Programming Publications, Dept. 452, San Jose, Calif. 95114

Page 9, Col. 2, Change to read as follows:
Line 19. P is from 1 to 32767.
Line 24. E = 1 through 32766. Replace the lowest
Line 31. E = 32767. Execute restart core load.

Page 10, Figure 6, Changes the first two lines of block X to read:

```
CALL UNQ (M, 20)
CALL UNQ (R, 20)
```

Col. 1. Change last paragraph to read:
When the CALL VIAQ statement is executed and there are entries in the queue, the highest priority entry is removed and used to call the core load it references.

Page 11, Col. 2, Change to read as follows:
Line 14. E = 1 through 32766. Replace the lowest
Line 18. E = 32767. Execute restart core load.

Page 19, Col. 2, Change to read as follows:
Line 1. ~~1. NAME is a skeleton subroutine~~
Line 2. 2. NAME is a mainline subroutine, all

Page 21, Col. 1, Change the first sentence of the last paragraph under "Programmed Timers" to read:
To provide the user with large time intervals, a larger time base can be specified for the programmed timers
Col. 2. Change to read:
Line 37 4 CALL COUNT (31, 4, 8*IHOUR)

Page 47, Col. 1. Change Line 5 under "File Protection" to read as follows:
about to write, this control can be achieved.

Page 48, Col. 2, Change the first sentence to read as follows:

NOTE: During a multiple sector write operation, the subroutine supplies the sector identification word as each sector is written.

Change line 6 under "Test Option" to read:

parameter is temporarily altered by DISKN during

Page 52, Col. 1, Change to read as follows:

		Hexadecimal	Required
Line 1:	Function	Value	Parameters*
Line 16:	Reset	A	Control

Page 53, Table 5. Change Line 1 under "Contents of A-register" to read: 0600.

Page 54, Col. 2, Change line 11 under "Buffering of Messages to Disk" to read:
message) has been completed, the longest stored (time-wise)

Page 57, Col. 1, Under "Operator Request Function", add the following:
An 1816 keyboard entry should be dependent on an interrupt from the keyboard request key. The 1816 is not considered a process I/O device.

Under "Non-Buffering of Messages to Disk", change to read as follows:

Line 1.	There is no maximum to the message unit size specified.
Line 3.	minimum size is 80 words.

Page 60, Col. 1, Delete lines 6 and 7 under "Read Sequential-Program Control".
Col. 2, Change to read as follows:

Line 7.	Data-Channel Addressing Mode
Line 9.	dressing options available. The addressing mode digit is used only if the function is read data channel.

Page 83, Col. 2, Change to read as follows:

Line 2.	0 - Standard data resolution
Line 3.	1 - Read without multiplexor overlap with
Line 5.	0 - No external synchronization; no continuous
Line 7.	1 - Second ADC

Page 84, Col. 1, Change to read as follows:

Line 19.	(0) is zero as a dummy address since there is no multiplexor overlap.

Col. 2, Change last two lines under "Example 4" to read:
IN3(1), IN3(23), IM(1), IM(42), IM(1), IM(42),
IM(1), IM(42), 0, LIM)

Page 104, Col. 2, Change to read as follows:

Line 3.	8 bits represent the characteristic. The mantissa is normalized to fractional form, i.e., the implied decimal point is between bits zero and one.

Page 106, Col. 1, Change line 1 under "Store FAC" to read:
LIBF FSTO, FSTOX, ESTO, or ESTOX

Page 108, Col. 1, Delete line 2 (DC ARG) under "Fixed-Point Fractional Multiply (short)

Col. 2, Change text under "Integer Absolute Value" to read as follows:

```
CALL        IABS
DC          ARG
Input       An integer in ARG
Result      Absolute value of (ARG) replaces (A-register).
```

Page 109, Col. 1, Change the last sentence of the second paragraph to read:
It is the user's responsibility to ensure that the indicators are reset before they are used.

Page 112, Col. 1, Line 11 Change to read as follows:
$a_5z^8 + a_6z^{10})$

Page 122, Col. 1, Change A and B definitions under "Call Queue": to read as follows:

```
A    DC    1                  priority number.
         through
          32767
B    DC    1 - 32766          replace lowest priority entry.

            or

            0                 ignore the call.
          32767               restart
```

Change the A definition under "Call Unqueue:" to read as follows:

```
A    DC     1                 priority number.
         through
          32767
```

Col. 2, Change the D definition under "Service Recorded Interrupts" to read as follows:

```
D    DC    1-32766            replace lowest priority entry.
             or
             0                ignore the call.
           32767              restart
```

Page 132, Right-most column in table: delete each "or" and each two-digit number preceeding it. (For example "3C or 3E" becomes "3E".

File this Newsletter at the back of the manual. It will provide a reference to changes, a method of determining that all amendments have been received, and a check for determining if the manual contains the proper pages.

READER'S COMMENT FORM

IBM 1800 Time–Sharing Executive System Specifications

Form No. C26-5990-1

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? ________________________
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other ________________________

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

C26-5990-1

fold fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. 452

fold fold

IBM 1800 Printed in U.S.A. C26-5990-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601

C26-5990-1

IBM 1800 Printed U.S.A. C26-5990-1

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York