NASP–9229–06
NAS Enroute Stage A
Contract FA65WA–1395

NAS OPERATIONAL SUPPORT SYSTEM

USER'S MANUAL

NOSS (UTILITY) Monitor

Model A3d2.1

15 August 1974

This manual contains reference information for using the NAS Operational Support System (NOSS) (UTILITY) Monitor in its interface with other system subprograms.

These change pages update this document to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

NAS Programming
IBM Federal Systems Division
NAFEC, Atlantic City, New Jersey

CHANGE HISTORY

This (06 level) publication of the NOSS (UTILITY) Monitor User's Manual is current with the NOSS Composite Tape labeled N3D21x.

Level	Date	Change Proposal Number	Comment
01	1 October 1968		Original Publication
02	28 August 1970		Includes changes 1 through 7
03	15 April 1973		Update for A3d1.2
04	15 August 1973		Update for A3d2.0
05	24 May 1974		A3d2.1 Update
06	15 August 1974		A3d2.1 Update

x = latest tape level

PREFACE

This manual provides a general description of the IBM 9020 Utility Programming System and detailed reference information on the use of the Utility Monitor Subprogram (UMON). Section I is a summary, indicating the main features and objectives of the system. Section II is intended as a Utility Monitor operator's reference manual, presenting in convenient form information required to use the utility programming system. Section III is a description of the System Tape Generation procedures. The descriptions of each subprogram have been condensed from more detailed information given in the publications listed here. Although references are made to these publications where appropriate, the descriptions in Section I are complete enough to provide an understanding of the functions and the input/output options of each system component.

The completion of a basic programming course, or its equivalent in experience, is the prerequisite for using this manual. Additional information about the utility programming system and its languages (JOVIAL and Basic Assembly Language) is given in the following publications:

IBM 9020 Data Processing System, Principles of Operation, Form A22–6852.

IBM 9020 Data Processing System, Symbolic Program Tape Edit, (SPTEDT).

IBM 9020 Data Processing System, System Edit Subprogram, (SYSEDT).

IBM 9020 Data Processing System, Compool Edit Program, (CMPEDT).

IBM 9020 Data Processing System, Library Edit Program, (LIBEDT).

IBM 9020 Data Processing System, Library Subroutines, (LIBARY).

IBM 9020 Data Processing System Loader Program, (LOADER).

IBM 9020 Data Processing System, Basic Assembly Language, (BALASM).

IBM 9020 JOVIAL Language User's Manual, (JOVIAL).

Text references to the preceding programming documents include subprogram or program. These differences will be resolved in a future republication.

This publication includes information presented in the *IBM 9020 Data Processing System, System Reference Manual,* Form Number C28–6549 and in the *User's Reference Manual for NOSS Monitor and Basic Processors* (Preliminary). The System SRL was provided under Contract FA64WA–5223; the NOSS Reference Manual was provided under Contract FA65WA–1395. Both manuals are superseded by this manual.

These change pages update the NAS Operational Support System (NOSS) User's Manual for the NOSS (Utility) Monitor, dated 24 May 1974 to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

NAS Programming
IBM Federal Systems Division
NAFEC, Atlantic City, New Jersey

CONTENTS

ILLUSTRATIONS

TABLES

1.0 THE UTILITY SYSTEM

1.1 SYSTEM SUMMARY

The IBM 9020 Utility Programming System is a comprehensive, integrated set of subprograms designed to facilitate program preparation and testing. It provides both a JOVIAL compiler and an assembly program permitting the programmer to write in either JOVIAL or Basic Assembly Language (BAL) or, subject to certain restrictions, in BAL within JOVIAL programs (direct code). The utility programming system also contains a loader program, which allocates 9020 storage and calls in routines from the library (supplied with the system); a debugging system for facilitating checkout; and a set of input/output routines. Four edit programs are supplied to provide maintenance of various system components. The utility programming system operates under control of the utility monitor, which allows the processing of stacked JOVIAL and/or BAL jobs. JOVIAL, BAL, and object programs may appear in any order within a job.

The utility programming system shown in Figure 1–1 is supplied in the form of a reel of magnetic tape (the system tape), which contains all of the components. The system edit program and the utility monitor allow for adding subprograms, replacing subprograms, and modifying any of the supplied subprograms on the system tape.

A JOVIAL library, supplied on magnetic tape, contains approximately 130 library routines. The programmer may add JOVIAL library routines, as well as BAL programs, to a library by using the library edit program.

The utility programming system is designed to operate on an IBM 9020 Data Processing System and requires a console typewriter, from two to ten (depending on the functions requested) magnetic tape units, and a minimum of 256K bytes of storage. Additional storage can be effectively utilized and a 128K byte version can be created without JOVIAL capability. The 1402 Card Read Punch and the 1403 Printer may be used optionally as the input and output units. The magnetic tape units required by the various system functions are shown in Figure 1–2. Input/output units not required by the utility system itself at object program execution time are available to the object program.

The utility programming system, as supplied, is designed to support the operation of the following input/output equipment:

IBM 2400 Series Magnetic Tape Units
IBM 1052 Console Typewriter
IBM 1402 Card Read Punch
IBM 1403 Printer

The utility monitor controls the operation of a single central processing unit. It attempts to provide maximum utilization of asynchronous elements, but is not designed to operate multiple programs in parallel. The selector channel (as well as multiplex channel) operations are scheduled for maximum efficiency by the utility monitor's input/output system.

The JOVIAL compiler, assembly subprogram, loader subprogram, and the edit subprograms are designed for maximum monitor independence and, with proper interfaces, may be used with 9020 monitors designed for different objectives.

All elements of the system are designed to facilitate the efficient use of the IBM 9020 Data Processing System by application programmers. The JOVIAL compiler produces efficient code from statements written in JOVIAL. The system components provide diagnostic listings, storage maps, program listings, etc., as appropriate. The programmer may request dumps; conditional dumps; or traces in a JOVIAL program, in an assembly language program, or at load time, with editing and listing provided in a variety of formats. All system components attempt to retain programmer-assigned names and programmer-designated storage allocations. For example, the loader attempts to load at the assembled address; since this is not always possible, storage dumps indicate both assembled locations and actual load locations.

Another aid provided for the programmer is a facility to create and maintain a file of card images on tape. This tape is called the Symbolic Program Tape (SPT) and is maintained by the SPT Edit Subprogram. The BATCH program is used to take card images from an SPT tape and create a .INPUT tape.

The JOVIAL compiler, the compool facility, and the JOVIAL library routines together provide for the preparation of very large programs by a team implementing a real-time application. The compiler can accept a JOVIAL program, get and utilize compool ("communications pool" – a collection of data definitions maintained as a magnetic tape file), and provide calls for procedures defined within the JOVIAL program being compiled and for library routines previously compiled and placed on the library tape.

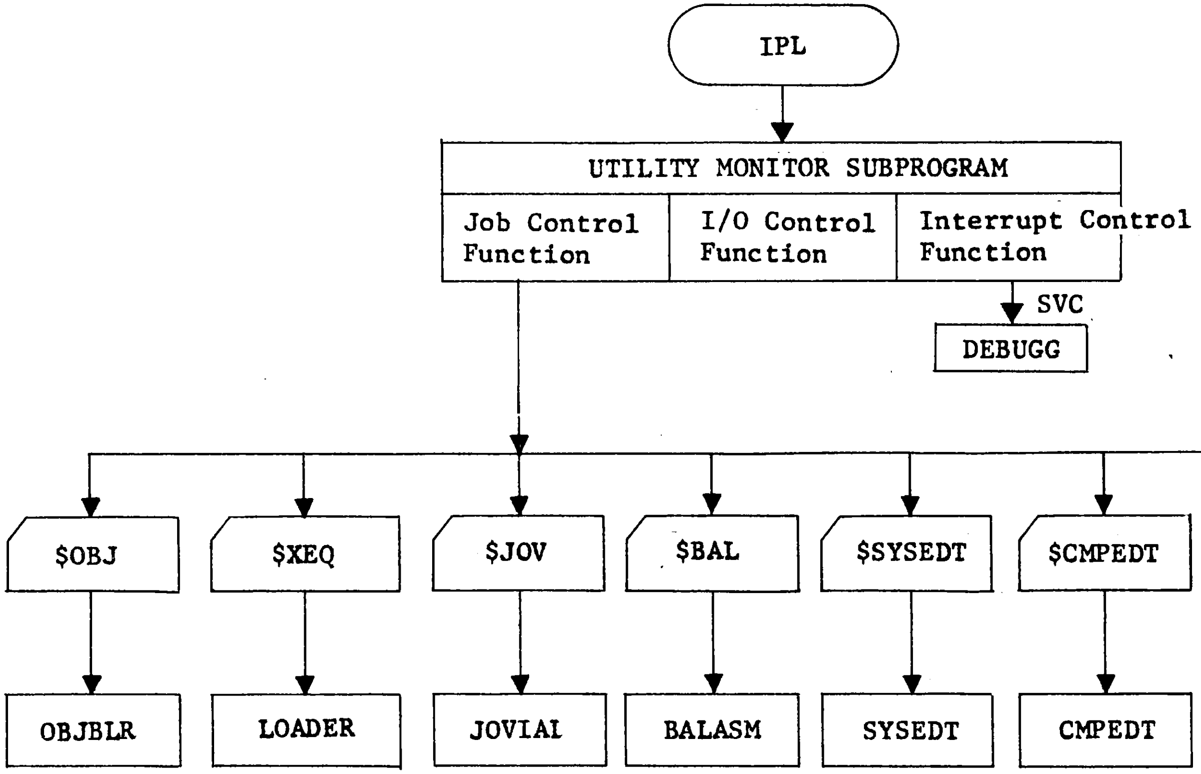


Figure 1–1. The Utility Programming System For The IBM 9020 Data Processing System

	.SYSTM	.INPUT[1]	.OUTPT[2]	.AUXIL	.LIB1	.COMP	.SPT1	.WORK1	.WORK2	.WORK3	Other	TOTAL[3]
Compilation	X	X	X	X	X	0		X	X	0[7]		9
Assembly	X	X	X	X			0	0[4]	0			7
Loading	X	X	X	X	0				0[8]			6
System Edit	X	X	X	X	0[5]				X			6
Compool Edit	X	X	X	X	0[5]	0[5]			X			6
Library Edit	X	X	X	X	0[5]				X			6
SPT Edit	X	X	X	X	0[5]		0[5]		X			6
Object Blocking Routine	X	X	X	X								4

Key:		
	X	Required
	0	Optional
	blank	Not needed

NOTES:

1. .INPUT may be either a tape unit or the on-line card reader.
2. .OUTPT may be either the .PRINT/.PUNCH tape or the on-line printer and card punch.
3. The total assumes that tape units are used for .INPUT and .OUTPT and includes any units shown as optional.
4. .WORK1 is used if the input to the assembler is a previously compiled JOVIAL source program.
5. An old master tape may be supplied for the edit programs, depending on the type of edit.
6. If a job includes compilation-assembly, and execution, with all options, 10 tapes are required.
7. Under certain conditions WORK3 is not needed for compilation.
8. WORK2 is required for loading if multiple pass executions are to be attempted.

Figure 1–2. Minimum Tape Configuration For The IBM 9020 Utility Programming System

The loader program searches the library, completes cross references between all programs and library routines, allocates storage, relocates programs (if required), and positions the compool-defined data in storage.

While absolute loading is permitted, the system is designed to produce completely relocatable object programs. Therefore, absolute location reference is never necessary with the utility programming system, although the programmer may, if desired, partially control storage allocation.

In providing for stacked-job operation, jobs that include problem-program execution may be interspersed between jobs that require only compilation and/or assembly. The system pre-empts portions of storage for the following purposes: resident monitor (including input/output elements and certain debugging control facilities); resident loader; LISTIO's (control blocks for programmer-designated input/output units); and user assignment table, reel table (if $REEL cards are present), and RLD information, from loader text, necessary for loading. All other parts of storage are available to the user's program. If he requests buffered input/output operation, the buffer area is constructed from storage not otherwise allocated. The allocation of core storage is illustrated in Figures 1–3 and 1–4.

The storage-protection feature of the IBM 9020 Data Processing System is supported by the utility programming system. The utility monitor has a unique protection key; the programmer may establish other protection keys for any designated portions of his program.

When an interruption occurs, the utility monitor takes prespecified action. In certain cases, the programmer may affect the action by masking the interruption or by providing a special alternate interrupt-processing routine of his own.

The utility programming system provides a number of aids to facilitate the serviceability of its components. Among these aids are an emergency storage-dump facility, a trace program, and a block of storage (used by the utility monitor and the system components for communication, input/output, error checking, and processing and/or execution).

The utility monitor controls the system. The programmer indicates sequences of operations within a job, prepares control cards signifying desired options, and supplies data cards, if desired, along with the programs submitted for execution. After processing and/or execution under utility monitor control, listings, maps, dumps, etc., are returned to the programmer.

The operator interacts with the utility monitor in several ways. Operator action initiates the loading of the utility monitor itself. Reassignment of input/output units, detachment and attachment of units, mounting and alternate use of tape units, all involve messages to and from the utility monitor via the console typewriter. Furthermore, messages regarding the results of program, job, and batch processing are logged for the operator.

Figure 1–5 shows utility programming system interrelationships. Control cards that determine the system component required for a particular task are indicated in the diagram. Solid connectors indicate paths that are normally taken; dotted lines signify optional paths. Note that for the compool, library, and SPT tapes, the option is determined by the program that will used the information on the tape and not by the edit program.

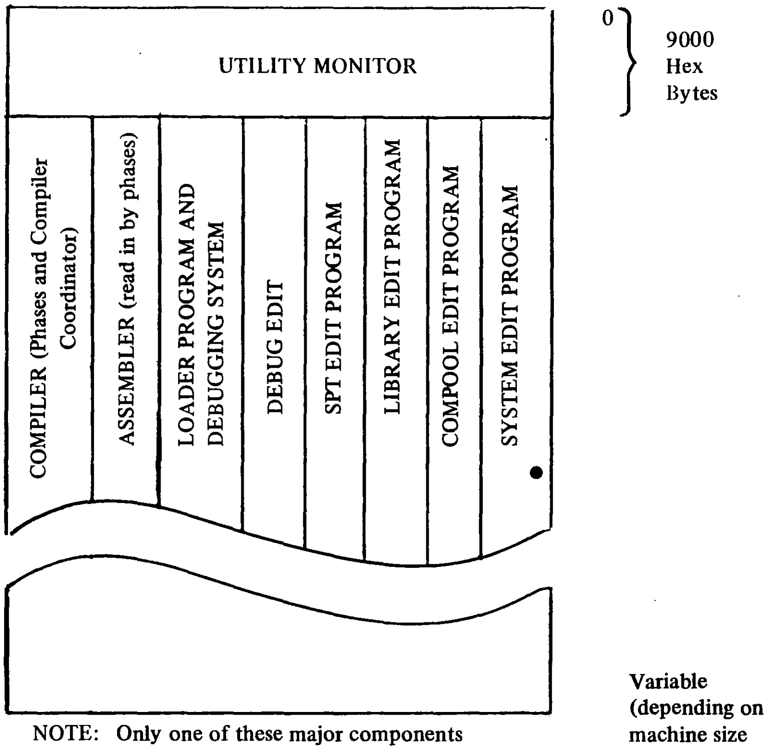
Flow of data through the utility programming system begins at the system input unit (referred to symbolically as .INPUT). This unit may be either a tape drive or the 1402 card reader; in either case, any system processor or the programmer may request card images with a calling sequence which is independent of the physical unit to be used. Similarly, the system output unit (.OUTPT) accommodates listing and punched card output, which may be placed on a tape unit (for peripheral processing) or on the 1402 punch and 1403 printer. The operator of the 9020 system specifies which physical devices (tape drives or 1402/1403) are actually used in a particular batch of jobs.

1.2 COMPONENTS OF THE UTILITY PROGRAMMING SYSTEM

The utility programming system consists of the following major components, all included on the system tape: utility monitor, JOVIAL compiler, assembler, loader, debugging system, and four edit programs. The library, supplied with the system, is recorded on the library tape.

The utility monitor (made up of job control, interruption control, an input/output system, and a communications region) is the system supervisor. It regulates all input and output, calls other components as required, permits communications among system components, and governs all operations.

The compiler processes source programs written in JOVIAL to produce output in Basic Assembly Language (BAL) that can be immediately submitted to the assembler. Associated with the compiler is the compool tape, prepared and maintained by the Compool Edit program. This tape is used to record compools, consisting of collections of data definitions that can be used by a team of JOVIAL programmers.



NOTE: Only one of these major components will be in storage at any specified time.

Figure 1–3. Approximate Storage Allocation Of Major System Components

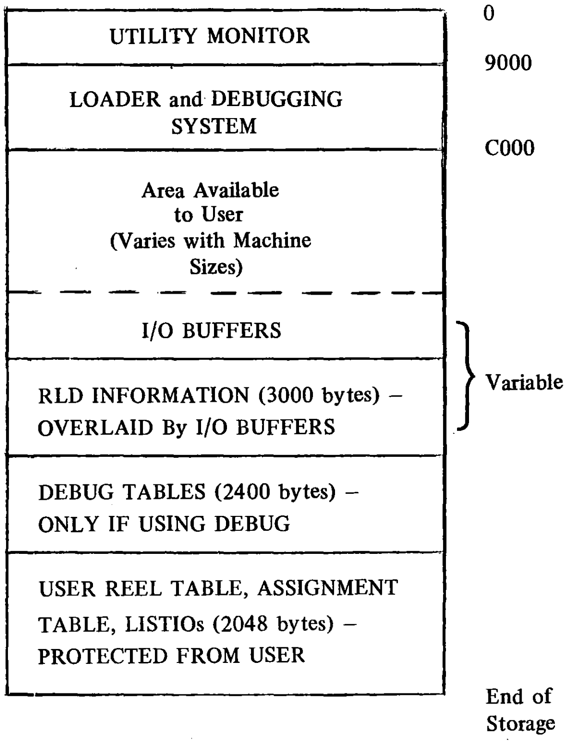
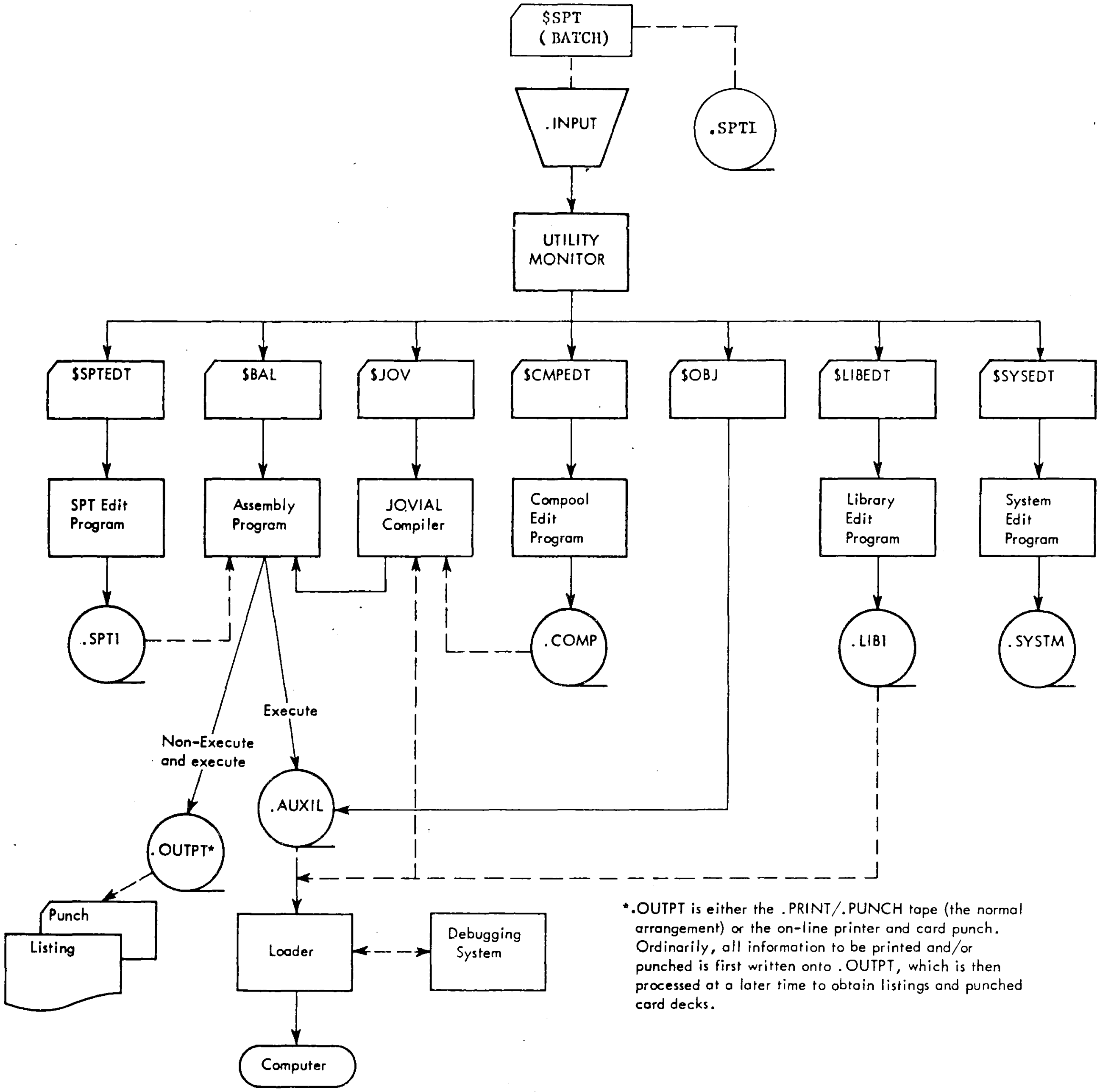

UTILITY MONITOR
0
9000
LOADER and DEBUGGING SYSTEM
C000
Area Available to User (Varies with Machine Sizes)
I/O BUFFERS
RLD INFORMATION (3000 bytes) – OVERLAID By I/O BUFFERS
Variable
DEBUG TABLES (2400 bytes) – ONLY IF USING DEBUG
USER REEL TABLE, ASSIGNMENT TABLE, LISTIOs (2048 bytes) – PROTECTED FROM USER
End of
Storage

Figure 1–4. Approximate Storage Allocation At Load Time



*.OUTPT is either the .PRINT/.PUNCH tape (the normal arrangement) or the on-line printer and card punch. Ordinarily, all information to be printed and/or punched is first written onto .OUTPT, which is then processed at a later time to obtain listings and punched card decks.

Figure 1–5. Relationship of Utility Programming System Components

The assembler translates BAL programs into loader language. Its output may be either printed and/or punched for later use, or submitted immediately to the loader for loading into storage and subsequent execution. The symbolic program tape (SPT), maintained by the SPT edit program, is used by the assembler. This tape can be used to store BAL programs.

The loader places assembled programs in storage for execution. In doing this it allocates storage, adjusts the assembler-assigned addresses, if necessary, prints out a map of the locations used, and loads the programs.

The debugging system provides for planned dumps and traces during program execution, for tape dumps, and for emergency storage dumps when a program cannot be successfully completed. Debug requests may be submitted with JOVIAL, BAL, or loader language card decks.

The library system includes the library edit program (used to add, delete, or insert routines in the library) and the library itself. The latter consists of approximately 130 support routines supplied with the system.

The system edit program is used to maintain the system tape, which contains all components of the utility programming system. It can add, delete, replace, or patch system components.

Figure 1–6 summarizes the operations performed by the system components, except for the utility monitor, which is described in Section 2.

1.2.1 UMON – Utility Monitor Subprogram

The Utility Monitor Subprogram (UMON) gains control of a computing element when an operator executes an Initial Program Load (IPL) procedure with a Utility Program System Tape (.SYSTM) mounted on the selected subchannel. UMON retains control until it identifies a requirement for other subprograms to operate. After transferring control to another subprogram, UMON regains control as the result of a computing element interruption.

When UMON gains control from an IPL procedure, it defines the IBM 9020 configuration for the program, provides appropriate initialization and operator communication, and examines .INPUT to determine the next job to be performed.

The operations performed by UMON as the result of a computing element interruption depend upon the type of interruption and the sense and status data associated with the interruption. The operations associated with each type of interruption are briefly described. Following these operations, control is normally returned to the interrupted subprograms or the next subprogram required as indicated by .INPUT.

The Supervisor Call instruction (SVC) interruption provides subprogram access to monitor routines. These routines are described in Section 2.0 and deal with the utilization of input/output, computing, and storage resources.

The Program interruption identifies potential error conditions in subprograms. UMON determines if the interrupted subprogram has previously designated (via appropriate SVC) a return control point for the specific program interruption and transfers as appropriate. When no return control is specified, either the current job is terminated or all program operations are terminated depending on error type.

The Input/Output interruption provides UMON with sense and status data relating to I/O subchannels. UMON takes appropriate action to update subchannel availability, initiate pending input/output operations for the subchannel, update completed input/output operation status, and retry I/O operations when error conditions are identified.

The External interruption provides UMON with signal indications from CCC hardware elements in the system, the interval timer, and the operator. UMON provides appropriate system termination service for error signals, updates the interval timer and responds to operator signal by alerting the operator to enter his request via the system typewriter.

The Machine Check interruption provides UMON with an indication of hardware detected computing element errors. UMON terminates program operation for this interruption.

Where operations must be terminated as the result of errors, UMON provides facilities to dump appropriate storage contents for subsequent analysis.

In addition to handling computing element interruptions, UMON provides, on request, a set of utility type services which perform card to tape, tape to card, tape to print, tape to tape, and tape comparison operations. These services are called the BATCH feature.

The inputs to UMON consist of the records on the utility program system tape, data stored in the communications region as the result of an interruption or subprogram operations, operator input messages on the system typewriter and programmer input card decks containing control cards, data, and code. The job control cards provide UMON with information that identifies each job in a batch and the boundaries of a batch job.

Operation	Control Card	System Component Called	Input	Intermediate Output	Component to which Intermediate Output is Submitted	Final Output
Compilation	$JOV	Compiler	JOVIAL program Compool Tape* Library Tape	BAL program on .WORK1	Assembly Program	JOVIAL program listing BAL punched deck*
Assembly	$BAL	Assembly Program	Compiler output or BAL program on SPT and/or system input	Object program on .AUXIL and .OUTPT (if $XEQ job); on .OUTPT only if $NONEX job	Loader (if $XEQ job)	From .OUTPT: BAL listing* Symbolic Analyzer listing* Punched object deck*
Loading	Operates only when $XEQ is in effect	Loader Program	Output of Assembly Program and/or object program(s) on .AUXIL. Library Tape*			Storage map. Object program(s) are loaded into storage for execution
Debugging	DBG cards for $XEQ only	Debugging System	DBG cards prepared by Assembly Program or by user. (Read in from .AUXIL by loader along with object program(s).)	Dump and/or trace on .AUXIL	Debug Edit	Dump and/or trace per DBG request
Compool Tape Edit	$CMPEDT	Compool edit Program	Old compool tape and/or mod definitions edit control cards: TERM statement.			Revised (or new) compool tape Listing
Library Tape Edit	$LIBEDT	Library Edit Program	Object programs and/or library tape; edit control cards.			Revised (or new) library tape. Punched object deck* List of contents
Symbolic Program Tape Edit	$SPTEDT	SPT Edit Program	BAL programs and/or statements and/or old SPT; edit control cards.			Revised (or new) SPT BAL punched deck* Listings* List of contents
System Tape Edit	$SYSEDT	System Edit Program	Old system tape; TXT and REP loader language cards; edit control cards.			Revised (or new) System Tape and List of Contents

* = optional

Figure 1–6. System Operations

The control cards indicate the extent and nature of jobs and batches to the monitor. The control cards indicate the beginning of each job in the batch, the type of job, the system capabilities required, the type of I/O units to be used, the presence of programmer-supplied data, the end of the job, and the end of the batch.

A note on the difference between jobs and batches: A job normally starts at a $ID card and ends at end of file on system input. The end of file may be simulated with a 7/8 EOF card. A batch starts with IPL and continues until the next IPL. The IPL may be performed manually from the console, requested via the typewriter, or induced by control cards on the system input unit.

All the control cards except 7/8 EOF must have a $ in column 1; all control cards except the comments card require a mnemonic starting in column 2. No embedded blanks are permitted. In the following control card descriptions, braces around entries indicate that one of the operations is to be selected; uppercase words and letters must appear exactly as given in the description; lowercase words and letters indicate the type of data to be supplied by the programmer.

The control cards fall into the following categories: job control cards, job-type control cards, capability control cards, I/O control cards, and the $DATA control card.

There are nine job control cards: $(comments), $PRINT, $ENV, $DATE, $PAUSE, $ID, 7/8EOF, $END, and $ENDTAPE.

The $DATE control card supplies the date for a batch of jobs. This card is supplied by the operator as the first card of the batch.

The $ID control card is required for proper identification of each job in a batch. For each job there must be one $ID control card, which must precede all control cards for the job.

The $ comments control card contains information that is to be placed on the system output unit and typed on the console typewriter. This control card is optional; if used, it must precede another monitor control card. The information on this card has no effect on the processing of the job. Columns 2 through 15 are blank. Comments start in column 16.

The $PRINT card is similar to the $ comment card except that output is to SYSOUT only.

The $PAUSE control card causes the monitor to enter a wait loop until the operator depresses the enter key on the console typewriter. The comments are placed on the system output unit and typed on the console typewriter; they may be used to provide operator instructions. This control card is optional; if used, it must precede another system control card.

The $ENV control card causes the monitor to verify that the job environment fulfills specified new requirements. This card is optional. If used, $ENV cards must follow the $ID cards and precede the job type control card. (When specifying several parameters, the user may optionally use more than one $ENV card to describe them.)

The $END control card indicates the end of a batch. This card precedes the 7/8EOF card of the last job for each batch on the system input unit except the last batch, which is terminated by a $ENDTAPE card. The operator may choose to terminate a batch after any job because of differences in job requirements.

The job type control cards indicate to the monitor the nature of the jobs that follow them. One of the following cards should follow a $ID or a $ENV control card:

- **$XEQ – Problem program is to be executed**
- **$AUX – Object decks, if any, are to be placed on the system auxiliary tape but are not to be executed as a problem program**
- **$NONEX – Problem program is not to be executed, and no system processor will read object decks**

If a $XEQ, $AUX, or $NONEX control card is used, it must be followed by one or more capability control cards, each of which may be followed by a source program or other deck according to individual processor requirements. These cards indicate to the monitor which capability is required by the job, as follows:

- **$BAL – BAL Assembly**
- **$JOV – JOVIAL Compilation**
- **$OBJ – Object Blocking (to copy the following object deck from the system input unit to the AUXIL tape)**

Input/Output control cards ($UNIT and $REEL) must be used to identify an I/O units (other than the system I/O units) used by a job. The monitor uses the information on these control cards to make unit assignments. As many of these cards as are needed may be used, but they must appear immediately after the last problem program deck for the job. Whenever possible, units are assigned to alternate channels in the order in which $UNIT control cards appear.

If data cards are used by the problem program in a job, the data deck must be preceded by the $DATA control card and must appear as the last physical portion of the job deck.

The outputs from UMON consist of data stored in the communications region, operator output messages on the system typewriter, programmer card decks on the .PUNCH unit, programmer listings on the .PRINT unit, and tapes resulting from card-to-tape and tape-to-card operations.

A detailed description of information required to use and operate the utility job control functions is provided in Section 2.

1.2.2 SYSEDT – System Edit Subprogram

The System Edit Subprogram (SYSEDT) is used to maintain the system tape (on .SYSTM) which contains all the subprograms of the Utility System. SYSEDT can replace, insert, or delete an entire subprogram record or patch portions of a record.

The input to the SYSEDT subprogram consists of control cards and input decks containing the subprograms and/or alterations to a subprogram, and the old master system tape. The control cards call in the system edit program ($SYSEDT) define the type of operations performed (INDEX, REPLACE, INSERT, DELETE, PATCH, and TM), identify the input deck that follows (TITLE), and indicate the end of the edit job (a $Control card).

The INDEX, REPLACE, PATCH, INSERT, TM, and DELETE cards follow the $SYSEDT monitor control card. They must appear before the first TITLE card and must be in the same order as the subprogram records on the tape. One TITLE card must precede the input deck for each subprogram record affected, except for a record that is to be deleted. The TITLE cards and the corresponding input decks must also be in the same order as the order of REPLACE, BATCH, and INSERT cards at the beginning of the deck. If not, the error messages are generated and the job is terminated. The input deck consists of loader text (TXT) cards and/or replace (REP) cards; all other loader language cards are ignored.

The output from the SYSEDT subprogram includes a revised system tape, an optional listing index of the old system tape, (an index listing of the new system tape) and error messages.

A detailed description of information required to use and operate the system edit functions is provided in the *IBM 9020 Data Processing System, System Edit User's Manual.*

1.2.3 LIBEDT – Library Edit Subprogram

The Library Edit Subprogram (LIBEDT) gains control when the monitor (UMON) identifies a $LIBEDT control card. The LIBEDT subprogram is used to generate and maintain a library tape of routines which are available to JOVIAL and BAL programmers. The subprogram can add, delete, or insert routines or punch an object deck of any routine in the library. The LIBEDT subprogram manipulates entire routines only; parts of individual routines cannot be altered.

The input to the LIBEDT subprogram consists of control cards, object decks for new routines, and, optionally, the old library tape. The control cards (LPUN, LINS, LDEL) define the operation to be performed.

The outputs from the LIBEDT subprogram consists of a new library tape, object decks if requested by an LPUN card, diagnostic messages, and a listing of the library tape. The information listed for each program on the library tape includes identification, external symbol dictionary entries, and procedure description data.

A detailed description of user/operator instructions and procedures is given in the Library Edit Program User's Manual.

1.2.4 CMPEDT – Compool Edit Subprogram

The Compool Edit Subprogram (CMPEDT) gains control when the monitor (UMON) identifies a $CMPEDT control card. The CMPEDT subprogram is used to generate and maintain a compool tape. The subprogram can add, delete, and replace compools.

The input to the CMPEDT subprogram consists of control cards, compool definition cards, and the old compool tape. The control cards (.ADD and .DEL) define the operations to be performed, the control card (.END) defines the end of the control cards, and the control card (.SEG) identifies segments within a compool. The .ADD and .DEL control cards follow the $CMPEDT control card and may appear in any order preceding the .END control card. Any number of compool definitions may be made, but the data declarations must be preceded by an .ADD control card. The compool data must be in the same order as the first set of .ADD cards. A compool definition may consist of JOVIAL data declarations for arrays, tables and their strings and/or item definitions (with or without preset constants), and parameter items. Single items are not permitted. The last card of each compool description must be a JOVIAL TERM statement. A compool definition may also include EQUATE statements but all equated data must be in the same segment. The JOVIAL language coding conventions and rules for data declarations apply to compool descriptions. Any number of compool descriptions may be contained in a compool edit source deck and sequenced one after the other, the only limits being that no more than 100 compools are permitted on one tape, and the tape must be a single physical reel.

The output from the CMPEDT subprogram consists of diagnostic messages, a tape index, a summary of each compool on the new compool tape, and the new compool tape.

Note: In an R-type compool edit, the tape to be revised (master tape) may be either an old compool tape or an old MLC (Merged Library/Compool) tape. In the latter case, an MLC tape is also created as output with the desired compool changes made.

Label checking still refers to the old compool tape label, as it is not altered when an MLC is created.

A detailed description of user/operator instructions and procedures is given in the *IBM 9020 Data Processing System, Compool Edit Program User's Manual.*

1.2.5 SPTEDT – Symbolic Program Tape Edit Subprogram

The Symbolic Program Tape Edit Program (SPTEDT) creates and maintains symbolic program tapes (SPT) The SPTEDT subprogram can add programs to or delete programs from the SPT and can also alter decks already recorded on the SPT.

The SPTEDT program is a single, self-contained program that builds a new SPT from an input of control cards, source cards, and correction cards. Any programs on the old SPT that are not deleted or revised are copied onto the new SPT prior to the conclusion of the edit. The SPTEDT subprogram produces a dictionary of all programs on the new SPT and, optionally, a source program listing and/or punched card deck. Diagnostic messages are issued when errors are discovered during processing.

The $SPTEDT control card initiates the program. Additional control cards are used to identify new programs to be added to the SPT, old programs to be altered, or old programs to be deleted from the old SPT. Changes to a single program must be ordered according to sequence numbers.

The SPT edit program prints a listing of any new or revised decks. It will assign new sequence numbers to decks being updated or added unless requested not to do so.

When the SPT is created or revised, the SPT edit program prints a list of the programs on the SPT in the same order as the programs are recorded. This list includes the names of any programs that were transferred unchanged.

The input units used are the system input unit and the old symbolic program tape (optionally).

Outputs from the SPTEDT subprograms are SPTs, printed listings, punched cards, and diagnostics.

Further details may be found in the *IBM 9020 Data Processing System, Symbolic Program Tape Edit User's Manual.*

1.2.6 JOVIAL – JOVIAL Compiler Subprogram

The JOVIAL subprogram is called by UMON when a $JOV control card is encountered on the system input unit. The major function of the subprogram is to translate JOVIAL source programs into Basic Assembly Language for subsequent processing by the assembler. During compilation, the subprogram checks all JOVIAL source program statements and issues diagnostic messages when errors are detected.

When library routines are requested in the source program the compiler sets up the required BAL calling sequence within the compiled program to call the library routine at execution time.

If a compool is requested, the compiler reads the compool from the compool tape. It writes the BAL DSs and DCs and descriptive table information on .WORK1 for BALASM and uses the table information to generate BAL operative code. Only one compool is used in a job. A detailed description of the JOVIAL language is given in the *IBM 9020 JOVIAL Language User's Manual.*

Programs written in the JOVIAL language are the primary input to the compiler. A $JOV control card must appear in front of every JOVIAL source deck submitted for compilation. Options of the control card include LIST, PUNCH and ANALYZ, which instruct the BALASM subprogram to list the BAL source program produced by the compiler, punch an object deck, and print a cross-reference listing of all symbols in the program. The PUNCHS option instructs the compiler to punch a BAL source deck. Other options are described in the JOVIAL User's Manual.

Within each source deck, the first card of the program must be a START card, and the last statement in the program must be a TERM statement.

Output from the compiler includes the BAL program, the JOVIAL source program listing and diagnostic messages reflecting conditions occurring during compilation, and the BAL source deck if the PUNCHS option was specified.

1.2.7 DEBUGG -- Debug Subprogram

The debug subprogram gains control when the monitor identifies that a debug request has been encountered via supervisor call in the program instruction stream being executed, and at the end of program execution if dump or trace information has been written on the auxiliary tape. The debug subprogram identifies and saves dump and trace information during program execution and edits and prints the information on completion of program execution.

Debug requests may be inserted for:

- **Trace information on each successful branch within the trace area**
- **Dump information each time specified instruction addresses are reached or specified conditions are met**
- **Dump information if program is unable to continue**
- **Logical tape records at end of job**
- **Physical tape records at end of job**
- **Trace information after the execution of each instruction in the trace area**
- **Timing information for each instruction in the specified trace area**

Inputs to the debug subprogram consist of data stored in monitor tables by the loader subprogram which identify dump and trace requests.

Outputs from the debug subprogram consist of the edited dump and trace printouts of data saved during program execution.

1.2.8 BALASM – BAL Assembler Subprogram

The 9020 Assembler translates BAL input into a form acceptable to the Loader program. The assembler provides program linkage for programs separately assembled that are to be loaded and executed together. It also examines the BAL program and prints diagnostic messages when errors are found. Tentative storage addresses are assigned to all program statements and data which may serve as actual storage addresses, if available, at load time. The programmer may request the use of common storage through the assembler so that programs within a job can refer to the same data. (Although similar in purpose, this is completely separate from the JOVIAL compool.) The assembler accepts output from the JOVIAL compiler and SYMCOR BAL source programs from the system input unit, and BAL source programs previously recorded on the SPT.

The assembler is called by the monitor when the compiler has completed processing or when a BAL program (indicated by a $BAL control card) is detected on the system input unit. When assembly processing is completed, control is returned to the monitor.

Inputs to the BALASM subprogram include the $BAL control card which directs the monitor to transfer control to the assembler and the source code to be assembled. The $BAL card also allows output options such as program listings, a cross-referenced listing of all symbols to be produced, a punched object deck, and an additional deck containing all program references to the library routines and compool.

Outputs include a program listing and diagnostic messages (serious and warning) reflecting conditions occurring during assembly.

If loading and execution are to take place immediately following assembly (.XEQ), the object program is recorded on .AUXIL for submission to the Loader program. The object program supplies the loader with the following types of information:

- **Text of the program**
- **Values assigned by the assembler to the program name and to all symbols used or defined in another program to be executed at the same time**
- **The address constants that must be changed if the program is loaded at a location different than the one assigned by the assembler**

When requested on the $BAL control card or the $JOV control card, the assembler produces an additional object deck. This deck contains references to library routines and compool elements used in the problem program.

The assembler is called by the compiler when compilation is successful. The LIST, PUNCH, and ANALYZ options from $JOV card are passed to the assembler. Input in this case is expected by the assembler to come from .WORK1.

SYMCOR output similarly appears on .WORK1 for input to the assembler.

The assembler (BALASM) and BATCH are the only subprograms in the system that allow direct SPT input.

1.2.9 LOADER – Loader Subprogram

The loader subprogram gains control when the monitor identifies a $XEQ control card and all input is on .AUXIL. The Loader program assigns storage addresses to the programs, places them in storage, and prints a record of the locations used. As soon as loading is complete, control is returned to UMON.

Historically, loaders serve a dual purpose: to get programs into memory from some other storage medium, and to resolve any problems caused by relocation (load addresses different than assembled addresses).

The use of base registers in the 9020 neatly solves much of the problem of relocation. However, a certain class of data, namely address constants, must be rectified on the basis of final physical load locations.

Input to the loader consists of twelve types of control cards, five of which are generated by the BALASM subprograms. The remaining seven are supplied by the programmer.

The assembler-produced cards (ESD, TXT, DBG, RLD, and END) provide the loader with the information required to properly locate and relate loaded programs. The ESD card provides information concerning the reference by one program to a symbol defined in another program. The TXT card contains the actual text of the object program. The DBG card contains information concerning dumps and traces. The RLD card provides information of address constants that must be modified if a program is relocated. The END card marks the conclusion of a program within a job and may designate the first executable instruction of the program (transfer address).

The programmer-produced control cards (REP, ORG, DBG, COM, TOV, LDT, and PUN) provide load-time options. The REP card alters an instruction or constant on a TXT card. The ORG card specifies the desired starting location of a program. The DBG cards provide debugging information. The COM card specifies the starting location of common storage. The TOV card allows segmented loading and execution of a program. The LDT card designates the address of the first executable instruction and signifies the end of the program deck for a job. The PUN card signifies that an absolute deck is to be punched.

The Loader output consists of a storage map listing, diagnostic messages data for the DEBUGG subprogram, punched object decks, or loaded subprograms.

A detailed description of the information and procedures required for the loading capability is given in the *IBM 9020 Data Processing System, Loader Program User's Manual.*

1.3 SYSTEM PROCESSOR VS. PROBLEM PROGRAM

The 9020 Utility System provides services for two types of subprograms: system processors and problem programs. A system processor is a subprogram resident on the utility system tape (.SYSTM). A problem program is any program which is not present on the utility system tape.

1.3.1 System Processors – General Information

System processors reside on the system tape in absolute format (i.e., the object code for the processor exists on the tape in exactly the same way as it will appear in storage when the processor is loaded). For this reason there are no special loading problems and the utility monitor may read a processor directly from the system tape into storage.

Monitor control cards inform the utility monitor of which processors are to be called. For example, the $JOV card causes the monitor to read the first phase of the JOVIAL compiler from the system tape and pass control to it.

1.3.2 Problem Programs – General Information

The user may select one of four operative procedures when working with a problem program:

1. Compile – assemble – load – execute

2. Assemble – load – execute

3. Load – execute

4. Compile and/or assemble

If the problem program is to be loaded and executed (procedures 1, 2, and 3), the object code must first be placed on the .AUXIL tape. This is done by the assembler if the input is source input (procedures 1 and 2) or by the object blocking routine if the input is an object deck (procedure 3*).

Problem programs are relocatable (i.e., they may be loaded at a location other than where they are assembled). As suggested in section 1.2.9, the object code for a problem program on the .AUXIL tape may differ from the way it will appear in core. The utility loader is used to read problem programs from the .AUXIL tape. The loader resolves externs, loads the program and loads any library routines which the program references. Appendix A, Section A.2, "Tracing a Program through Compilation, Assembly, and Execution" gives a more detailed description of the load function.

1.3.3 Input/Output Characteristics of System Processors

The I/O environment for any system processor must be determined during the design phase when writing the processor. In other words, system processors require not only a certain number of I/O units, but they require specific logical I/O units. For example, the JOVIAL compiler requires at least the following: .SYSTEM, .AUXIL, .SYSIN (tape or card reader), SYSOUT (tape or printer/punch), .LIB, .WORK1, .WORK2, and .TYPE. This is not to imply that the compiler might not need other tape units. However, if one of these I/O units were not attached to the system in which the compiler was to operate, the compiler would have to stop execution because of insufficient I/O units.

System processors refer to I/O units by system defined logical unit numbers. These numbers and their corresponding units are:

1	–	**SYSTM**
2	–	**SYSIN**
3	–	**PRINT**
4	–	**PUNCH**
5	–	**AUXIL**
6	–	**.LIB**
7	–	**.COMP**
8	–	**.SPT1**
9	–	**.WORK1**
10	–	**.WORK2**
11	–	**.WORK3**
12	–	**.TYPE**

The numbers 1–23 may be used as system defined logical unit numbers. Numbers 18–23 are used only in special instances, e.g., when non-standard I/O devices are to be used.

After being read in from the system tape, a system processor must account for the presence of necessary I/O units. The units .SYSTM, SYSIN, SYSOUT (PRINT and PUNCH) and .AUXIL need not be accounted for since these units are always within the system environment. A system processor, coded in JOVIAL, can check for the presence of any I/O unit by using the GETLIO library routine. A system processor coded in BAL must either use the GETLIO library routine or check the unit assignment table in the utility monitor. If using the latter method, the system processor must inspect absolute address X(5FC) to obtain the starting address of the monitor's Unit Assignment Table. The table consists of 2-word entries, the first entry being associated with Logical Unit 1, the second entry with Logical Unit 2, and so on. Each entry has the following format:

Word 1	–	**Bytes 1 and 2**	–	**Logical unit number**
Word 1	–	**Bytes 3 and 4**	–	**Device address**
Word 2	–			**Address of associated LISTIO**

By checking bytes 1 and 2 of the desired entry, the user can determine assignment. If the logical unit number field of the entry equals zero, the device is not assigned; the presence of the logical unit number indicates assignment. If an I/O device required by the system processor's I/O environment is not assigned, the system processor must discontinue processing and return control to the utility monitor.

A system processor is coded so that it can define and re-define its tape unit record formats and buffering requirements during operation. This facility is unique to system processors and is performed via the SVC SYSDTF or by the SYSDTF library routine. A single SYSDTF informs the monitor of record formats and buffering requirements for a tape mounted on a particular tape unit.

System processors are responsible for issuing tape mounting and/or tape saving instructions required during execution.

1.3.4 Input/Output Characteristics of Problem Programs

A problem program must inform the utility monitor of the number and types of I/O units to be used. However, the problem program is not concerned with which system-defined logical units are attached to the system. It is only

*The object blocking routine is a system processor whose sole purpose is to place object decks on the .AUXIL tape. The monitor calls the object blocking routine when a $OBJ card is read. See Section 2.4.7.

concerned with the number of units attached. For this reason the problem program does not use the system defined logical unit numbers (1–17), but instead uses logical unit numbers 20–99. For each logical unit number the problem program references, a $UNIT monitor control card must be supplied.

A $UNIT card identifies one logical unit number used in the program and specifies the device type (tape, reader, printer, or punch) referenced by the logical unit number. The $UNIT card also specifies the record formats and buffer requirements if the logical unit number refers to a tape.

A $REEL card may also be supplied with a $UNIT card. $REEL cards are supplied when mounting instructions are to be issued by the monitor for the first or only reel of a tape file. $UNIT and $REEL cards are described in detail in Section 2.2.5 of this manual.

In contrast to system processor, the problem program $UNIT and $REEL cards are evaluated prior to the problem program being loaded and executed. When the utility monitor encounters $UNIT and $REEL cards at the end of the input deck, it begins assigning available I/O units (except .SYSTM, .AUXIL, .SYSIN, and .SYSOUT). Any mounting or saving instructions as identified by $REEL cards are issued. If there are not enough I/O units to satisfy all $UNIT cards, the monitor will print an appropriate message and prevent the problem program from loading and executing. If enough I/O units are assigned, the utility loader loads the problem program and passes control to it.

1.3.5 Patching System Processors

Patches to system processors can be made via card or via IBM 1052 typewriter keyboard. System processors comprise one or more system records on the utility system tape. Separate procedures are used for patching processors one system record long and for patching processors that are contained in two or more system records.

The following procedure is used to patch a system processor one system record in length at load time:

a. Punch the character P in column 9 of the $ control card (monitor control card) which calls the system processor.

b. Put all REP cards directly behind the $ control card.

c. Put a card with a blank in column 1 and the characters BEGIN in columns 2 through 6 behind the REP cards.

For system processors exceeding one system record in length, a special REP card, the propagator, is required when patching any system record other than the first record. The format of the propagator REP card is:

12–2–9 punch in column 1
REP in columns 2 through 4
blank in columns 5 and 6
00052C in columns 7 through 12
blank in column 13
001D740 in columns 14 through 19

Each system record after the first record and preceding the record to be patched must contain a propagator card and a BEGIN card. The following procedure is used to patch system processors consisting of more than one system record at load time:

a. Perform all of the steps described previously for the first system record.

b. Put the propagator REP card and BEGIN card into the input stream for each system record preceding the record to be patched.

c. Put the REP cards containing the patch data and the BEGIN card into the input stream for the system record being changed.

REP cards (the propagator and patch cards) are placed in the input stream immediately preceding the phase of the system processor using those REP cards. For example, a programmer may wish to patch the second phase of a processor. If the first phase of this processor read all the input cards, the deck set-up would be as follows:

```
$ID
$NONEX
$XXXXXX  P  (processor control card with "P" in
             column 9)

12
 2
 9          REP 00052C  001D740
            BEGIN
            Input deck for processor
            REP cards to patch phase 2
            BEGIN
```

The following procedure is used to enter patches via the IBM 1052 typewriter keyboard during processor operation:

a. Depress the REQUEST key.

b. Type PATC on the 1052 typewriter keyboard.

c. Depress ENTER key and wait for CARDS response.

d. Type card images of REP, and/or PUT, and BEGIN cards.

   PUT is a special method of entering REP cards using the PATC option. The format is:

   ƀ PUT ƀ (6 character hex address) ƀ (hexadecimal data in contiguous byte strings).

e. Depress ENTER key.

Care must be taken when patching a system processor, that the processor which is being patched does not immediately follow a processor of more than one core load, all in the same job. This would cause the second system record of the first processor to be patched.

1.3.6 Patching Problem Programs

Patches to problem programs are also made via card or via IBM 1052 typewriter keyboard. The procedure used for patching problem programs after loading is the same as the procedure described for patching system processors using the IBM 1052 typewriter.

CAUTION

Problem programs are relocatable at load time. Patching the loaded program requires knowledge of absolute core addresses so that REP and PUT card images can correctly overlap the contents of those addresses.

1.3.7 Debugging Differences

System processors cannot use the services of the DEBUG subprogram. Once a subprogram has been cataloged to the utility system tape as a system processor, all DBG cards containing dump and trace information are ignored; only system dumps are allowed if processor operation cannot continue. The DEBUG subprogram overlays storage addresses occupied by the system loader, but the system loader does not load system processors. Therefore, no storage is available for DEBUG.

Problem programs can use the full facilities of the debugging system. Emergency dumps use information from the DUMPE control card. If no DUMPE card was included in the problem program, a dump is given from address 0 to address X'1B9F'.

1.3.8 Converting a Problem Program to a System Processor

If the user anticipates that a problem program is to become a system processor, he can code and debug it as a system processor using the $ENV monitor control card. The MOD=SYST option must be selected so that the problem program can use those logical unit numbers reserved for system processors.

If the user has written a problem program and at a later time wishes to convert it to a system processor, the following steps should be taken:

a. Determine the symbolic I/O units required for the new processor's I/O environment and replace references to logical unit numbers 20–99 with references to logical unit numbers 0–12. (See note.)

b. Add the code to check the utility monitor's LISTIO's for I/O unit availability.

c. Add code to specify record formats and buffering requirements using SYSDTF.

d. Add code for tape mounting and saving messages.

e. Obtain an absolute deck for use in cataloging the subprogram to the utility system tape.

f. Perform the utility system tape generation procedures explained in Section 3.

   Note: When writing a problem program, a user can reference symbolic I/O units in the code and equate these symbolics via parameter items to logical unit numbers acceptable to problem programs. When converting a problem program to a system processor the user can remove these parameter items and supply a new set of parameter items referencing logical unit numbers used by system processors.

1.4 UTILITY LIBRARY

The Utility Library Tape contains library routines that can be requested by any JOVIAL or BAL source program.

Library routines are separately compiled routines that perform an operation needed by a number of programs. The library routines are accessed by a specific call and bound to the user's program at load time. Each library routine has its own unique call which is either a JOVIAL function or procedure call.

JOVIAL functions are defined procedures that produce one value each time they are executed. A library routine that is a function is referenced by a function call. A function call is not a complete operative statement and must be used in conjunction with an expression. An example of a library routine that is a function call is CLC (Compare Logical Character).

IF CLC(AAA,10,BB) NQ 0 $

The function call CLC is contained within the IF statement.

JOVIAL procedures are defined procedures that can produce several values each time they are executed. A library routine that is a procedure is referenced by a procedure call. A procedure call, unlike a function call, is a complete operative statement. An example of a library routine that is a procedure call is MVC (Move Characters).

MVC(DATAOT,15,DATAIN) $

All library routines are on the Utility Library Tape. The Library tape is formatted as follows: header record, procedure descriptor table, tapemark, external symbol dictionary table, and library routines and a tapemark. Each library routine is composed of an identification record and program records. Figure 1–7 illustrates the format of the library tape. The library edit program creates and updates the Library Tape. More information on library edit can be found in the Library Edit Subprogram User's Manual.

The information on the library tape is used by the JOVIAL compiler and the system loader. If the library call is coded in BAL, then only the system loader is used. If a program contains a JOVIAL library call, the JOVIAL compiler searches the Procedure Descriptor Table for an entry corresponding to the JOVIAL library call. When the entry is found, the compiler takes the parameter information and generates the BAL code to set up the linkage between user program and the library routine (see Figure 1–8).

The compiler generates the name of the library routine as an external symbol. If a library call is coded in BAL, the programmer must supply an extern for the library routine name. It isn't until load time that the external symbols are resolved.

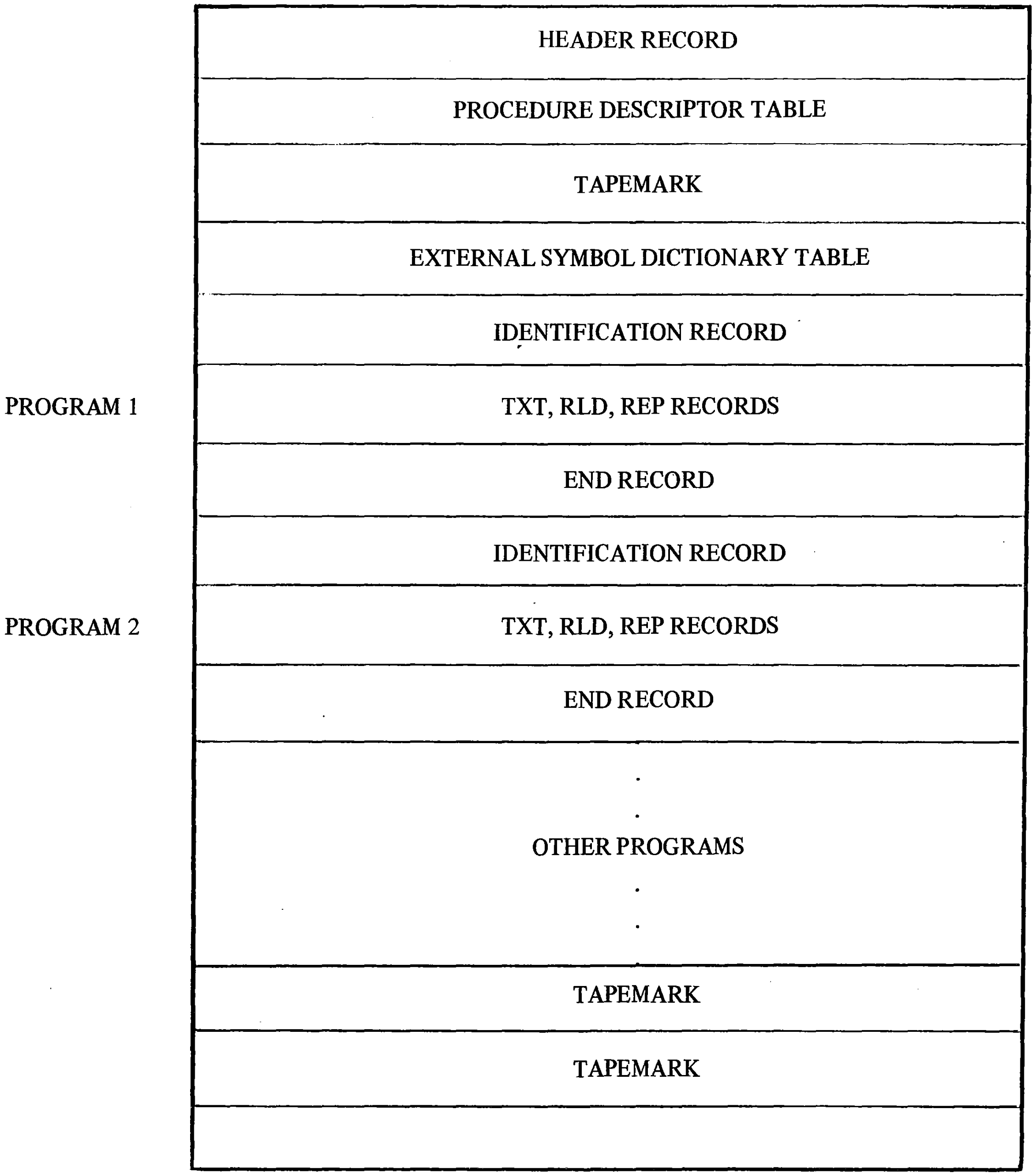
During the loading of a program, if the loader encounters an external symbol, the utility library tape is searched for a match in the external symbol table. If an entry is found for the symbol, the loader searches the tape for the library routine and loads the routine into storage. It then places the name of the routine and the load address in a table, which is used to resolve an address constant produced by the library call. When loading is completed, the library routine resides in memory and is bound to the user's program. (See Figure 1–9).

If the library routine is called in BAL, the programmer writes the parameter information that the JOVIAL compiler generates.

There are some library routines that are reentrant. A reentrant library routine is one which can be accessed simultaneously by programs operating in different computing elements. The reentrant library routines are coded, such that the data operated upon is within the user's program. Thus, input and output of the library routine is contained within the user's program and the problem of data being destroyed is eliminated.

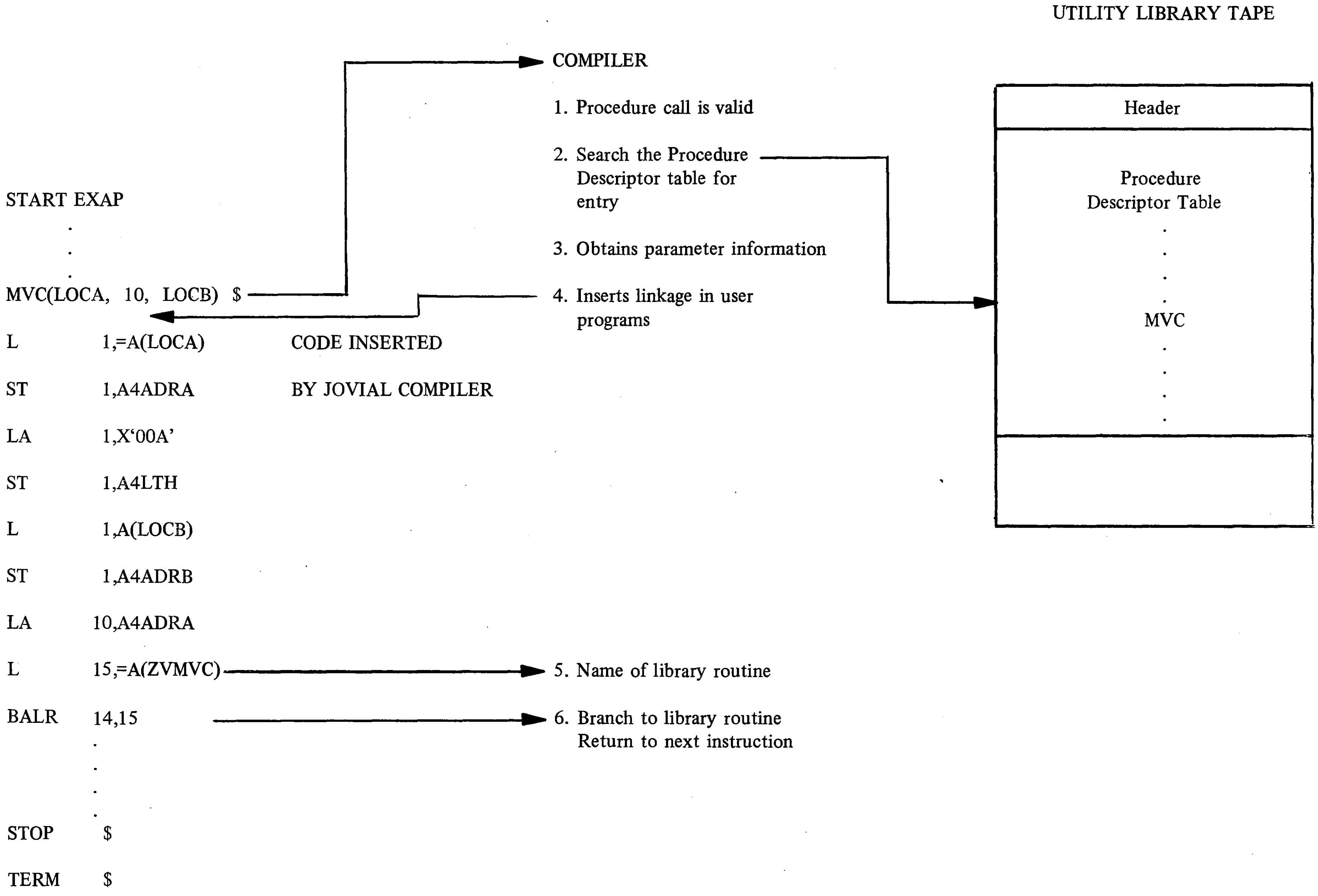
Figure 1–10 illustrates the interaction of Library tape with the Utility Programming system.

For more detailed information on the library, consult the Library Subroutines User's Manual.



	HEADER RECORD
	PROCEDURE DESCRIPTOR TABLE
	TAPEMARK
	EXTERNAL SYMBOL DICTIONARY TABLE
	IDENTIFICATION RECORD
PROGRAM 1	TXT, RLD, REP RECORDS
	END RECORD
	IDENTIFICATION RECORD
PROGRAM 2	TXT, RLD, REP RECORDS
	END RECORD
	. . OTHER PROGRAMS . .
	TAPEMARK
	TAPEMARK

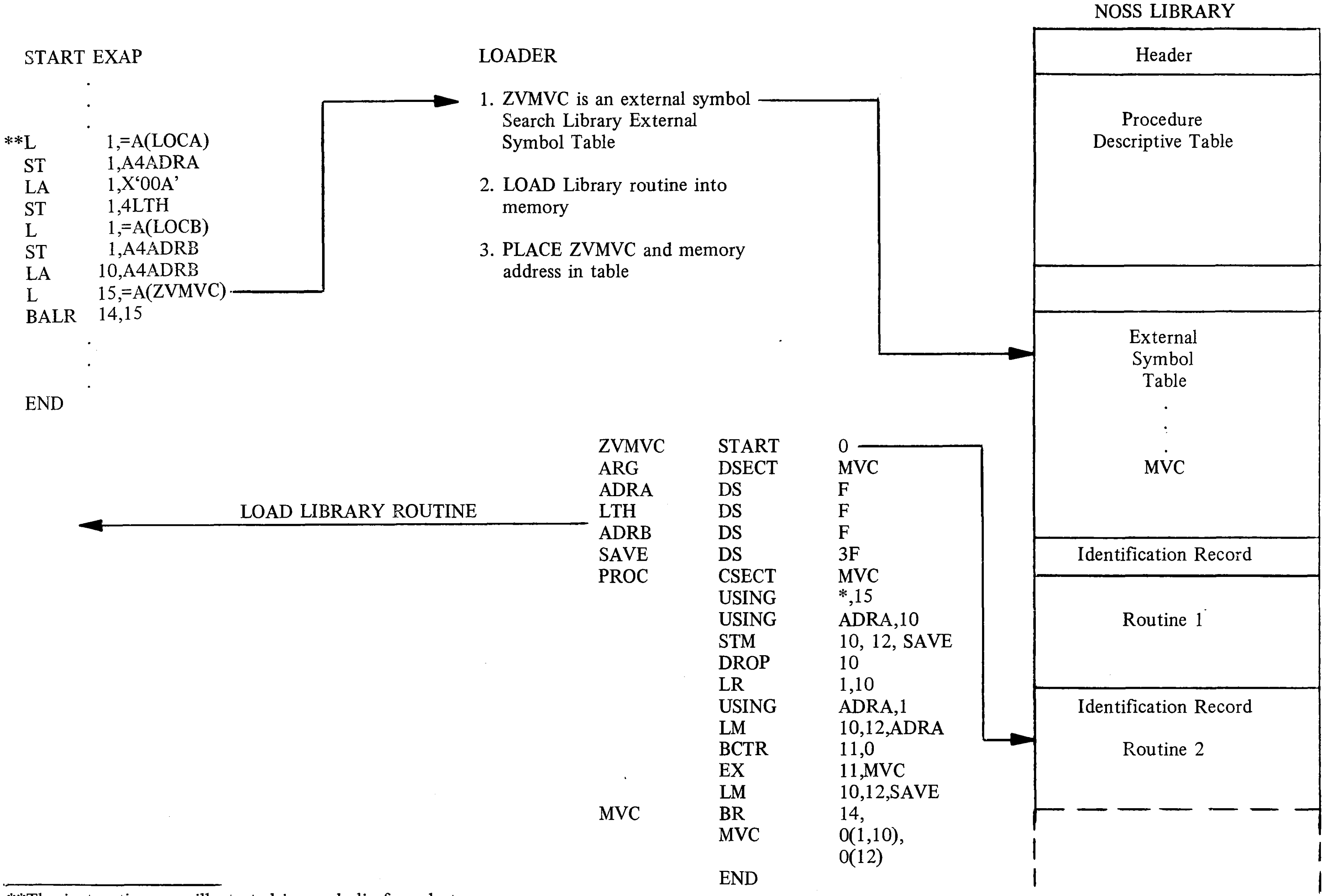
Figure 1–7. Library Tape Format



```
START EXAP
      .
      .
      .
MVC(LOCA, 10, LOCB) $
L          1,=A(LOCA)
ST         1,A4ADRA
LA         1,X‘00A’
ST         1,A4LTH
L          1,A(LOCB)
ST         1,A4ADRB
LA        10,A4ADRA
L         15,=A(ZVMVC)
BALR      14,15
           .
           .
           .
           .
STOP       $
TERM       $
```

1. Procedure call is valid
2. Search the Procedure Descriptor table for entry
3. Obtains parameter information
4. Inserts linkage in user programs
5. Name of library routine
6. Branch to library routine Return to next instruction

Figure 1–8. User Program – Library Routine Linkage



```
 START EXAP
          .
          .
          .
**L       1,=A(LOCA)
  ST      1,A4ADRA
  LA      1,X‘00A’
  ST      1,4LTH
  L       1,=A(LOCB)
  ST      1,A4ADRB
  LA      10,A4ADRB
  L       15,=A(ZVMVC)
  BALR    14,15
          .
          .
          .
  END
```

LOADER

1. ZVMVC is an external symbol Search Library External Symbol Table

2. LOAD Library routine into memory

3. PLACE ZVMVC and memory address in table

```
ZVMVC   START   0
ARG     DSECT   MVC
ADRA    DS      F
LTH     DS      F
ADRB    DS      F
SAVE    DS      3F
PROC    CSECT   MVC
        USING   *,15
        USING   ADRA,10
        STM     10, 12, SAVE
        DROP    10
        LR      1,10
        USING   ADRA,1
        LM      10,12,ADRA
        BCTR    11,0
        EX      11,MVC
        LM      10,12,SAVE
MVC     BR      14,
        MVC     0(1,10),
                0(12)
        END
```

**The instructions are illustrated in symbolic form but actually they are in machine language.

Figure 1–9. Library Routine At Load Time

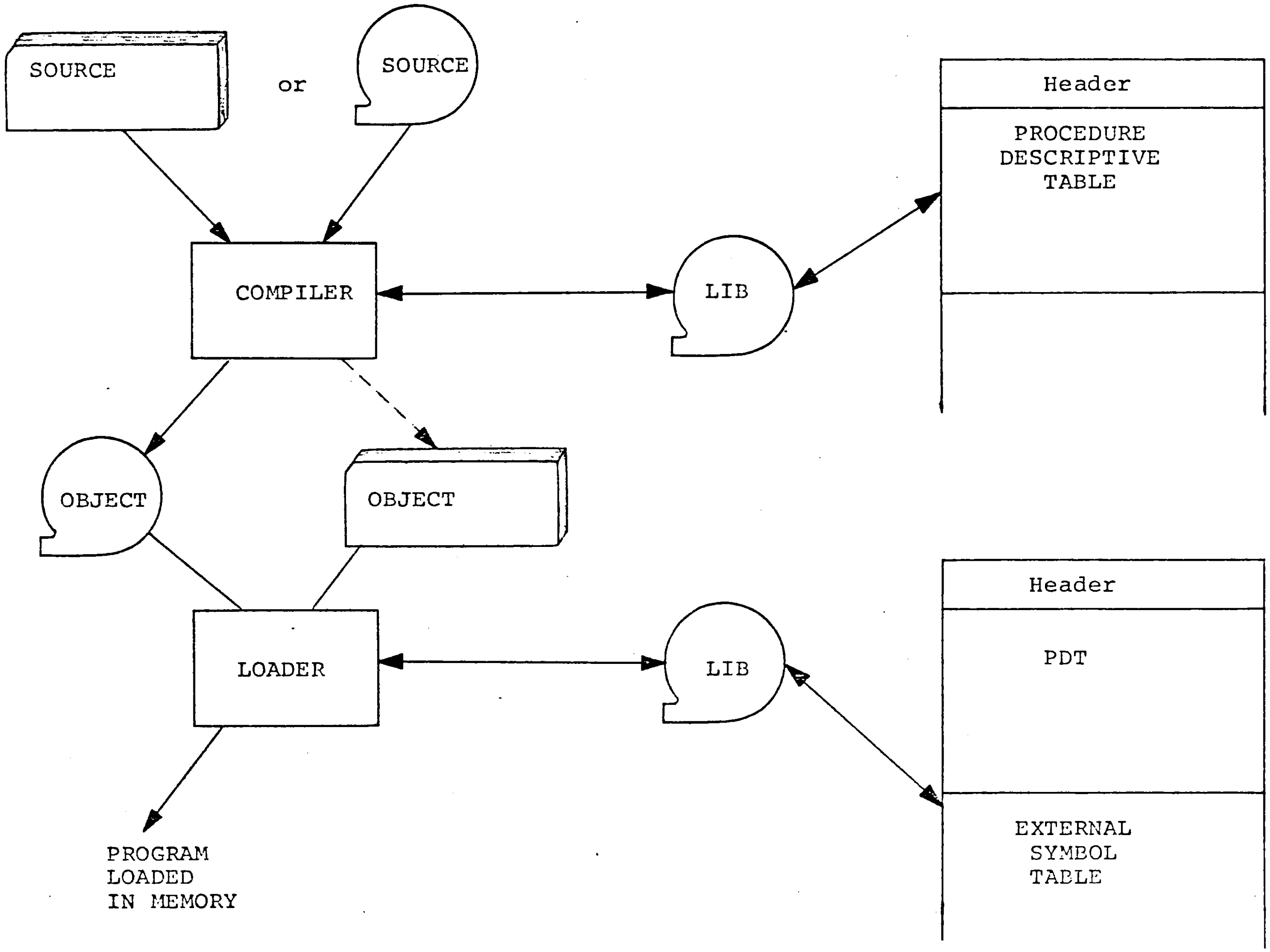


Figure 1–10. Library Tape – Utility System Relationship

2.0 SYSTEM CONTROL – UTILITY MONITOR

2.1 GENERAL INFORMATION

The utility monitor controls and supervises the operation of the Utility Support Program. It is the first processor loaded at IPL (Initial Program Load) and remains in storage through operation of all other processors. It regulates the loading of system processors, provides for communication between processors, handles all input and output devices supported by the system, controls interrupts, provides utility functions through supervisor calls, and issues status messages to the operator.

The monitor consists of five major sections:

1. Job Control. Governs control-card logic, I/O unit assignment, and the loading of system processors.
2. IOPACK. A collection of I/O routines that can be used by the programmer as well as by the system processors.
3. MONIO. Maintains asynchronous operations on all buffered units receiving requests through IOPACK.
4. Interrupt Control. Processes all interrupts.
5. Communications Region. Address constants, tables, flags, etc. used for communication within the monitor, between the monitor and processors, and between processors.

2.1.1 Use of This Section

The expected primary users of this section will be programmers with comparable backgrounds. The programs they produce, however, and the program requirements which must be met by the monitor, may show considerable diversity. There are two basic divisions: (1) JOVIAL versus BAL source code, and (2) problem programs versus system processors.

The first level of access to monitor services is via control cards. This section should be read by all.

Programmers coding problem programs in BAL need to understand IOPACK and non-I/O supervisor calls.

JOVIAL programs gain access to monitor services via library calls, which are described specifically in the Library User's Manual and generally in Section 1 of this manual.

The section on System Processors versus Problem Programs is intended for prospective additions to a system tape. Related to this is the section on "Overriding System Protection Features"; essentially, methods for allowing problem programs to run under system processor rules.

All users should understand at least the basics of the monitor sub-function BATCH.

The 9020 Utility System Program and the monitor provide powerful programming aids. They must be used properly. Read carefully. Any one can learn by mistakes. The efficient and productive programmer does it right the first time.

2.1.2 Initial Program Loading

The Initial Program Loading (IPL) procedure resides at the beginning of the system tape. To load the monitor into storage, the operator selects the tape drive on which the system tape is mounted with the loading switches on the system control panel and depresses the load button. The IPL procedure loads all the monitor components into storage. Once in storage, the monitor analyzes its environment, responds to operator messages, and assigns available I/O equipment.

2.1.3 System Protection

The monitor protects itself against destruction due to error in controlled programs, either system processors or problem programs. To do this, it operates in the supervisor state with a PSW protect key of zero, assigns a key of one to its own storage area, and assigns a key of two to the remainder of storage. When loaded, controlled programs are assigned a PSW key of two. Controlled programs may change the storage-protect keys for their own storage and may change their PSW key to any value except zero or one. This is done by supervisor calls.

The utility system processors also assume the right to a PSW key of zero.

In the USER Mode protection is suppressed and accidental or deliberate user destruction of the monitor can occur. The USER mode gives greater flexibility of operation but presents obvious risks. It is used only for a particular batch with operator knowledge.

2.1.4 Control Cards, Jobs, and Batches

A batch is started at IPL and continues until the next IPL; manual IPL was discussed previously. The IPL sequence may also be initiated from the typewriter or by cards.

A batch comprises one or more jobs. A job consists of one or more functions, such as assembly, loading, and execution. Jobs are defined and delimited by monitor control cards.

The various subprograms that may support a job are called by monitor control cards. As a rule, these subprograms require further control cards of their own. BAL assemblies, for example, require a "START" card.

Monitor control cards are covered in detail in this document. Subprogram control cards are discussed in their relevant System User's Manual.

Note the distinction between a batch of jobs and the monitor sub-function BATCH. The latter is a subprogram, described in paragraph 2.11. One of the functions of BATCH is to prepare batches.

2.2 MONITOR CONTROL CARDS

Normal sequencing and control of system functions is determined by control cards on the system input unit, .SYSIN. While .SYSIN may be a card reader or a tape, the data is always in card format.

All the control cards have a $ in column 1 and a mnemonic starting in column 2. There are two exceptions: the comments card and the 7/8EOF card. Other data on the card starts in column 10 for the $ID card and in column 16 for all others.

Monitor control cards are categorized:

a. Operator/Batch: Defines and delimits the batch.

b. Job Control: Defines and delimits the job and its environment.

c. Job-type: Indicates type of job (one per job).

d. Processor: Call a subprogram from the system tape.

e. Input/Output: Primarily defines tape.

f. $DATA: Indicates a data deck follows:

In the following control-card descriptions, braces around entries indicate that one of the options is to be selected; upper-case words and letters must appear exactly as given in the description; lower-case words and letters indicate the type of data to be supplied by the programmer.

Sample control cards are shown in Figure 2–1. Figure 2–2 illustrates a sample card deck for a 2-job batch. Figure 2–3 illustrates a sample job deck.

2.2.1 Operator/Batch Control Cards

There are four control cards that are normally supplied by the operator. They are:

$DATA – date card. Date is any eight characters starting in column 16. (nominally mm/dd/yy)

$END – End of batch is indicated.

$ENDTAPE – End of last batch on .SYSIN is indicated.

7/8EOF – End of job is indicated. This card is also used after $END and $ENDTAPE cards. Seven and eight are overpunched in column 1. This card simulated an end-of-file on SYSIN.

2.2.2 Job Control Cards

There are five job control cards: $ID, $ENV, $PAUSE, $(comments), $PRINT.

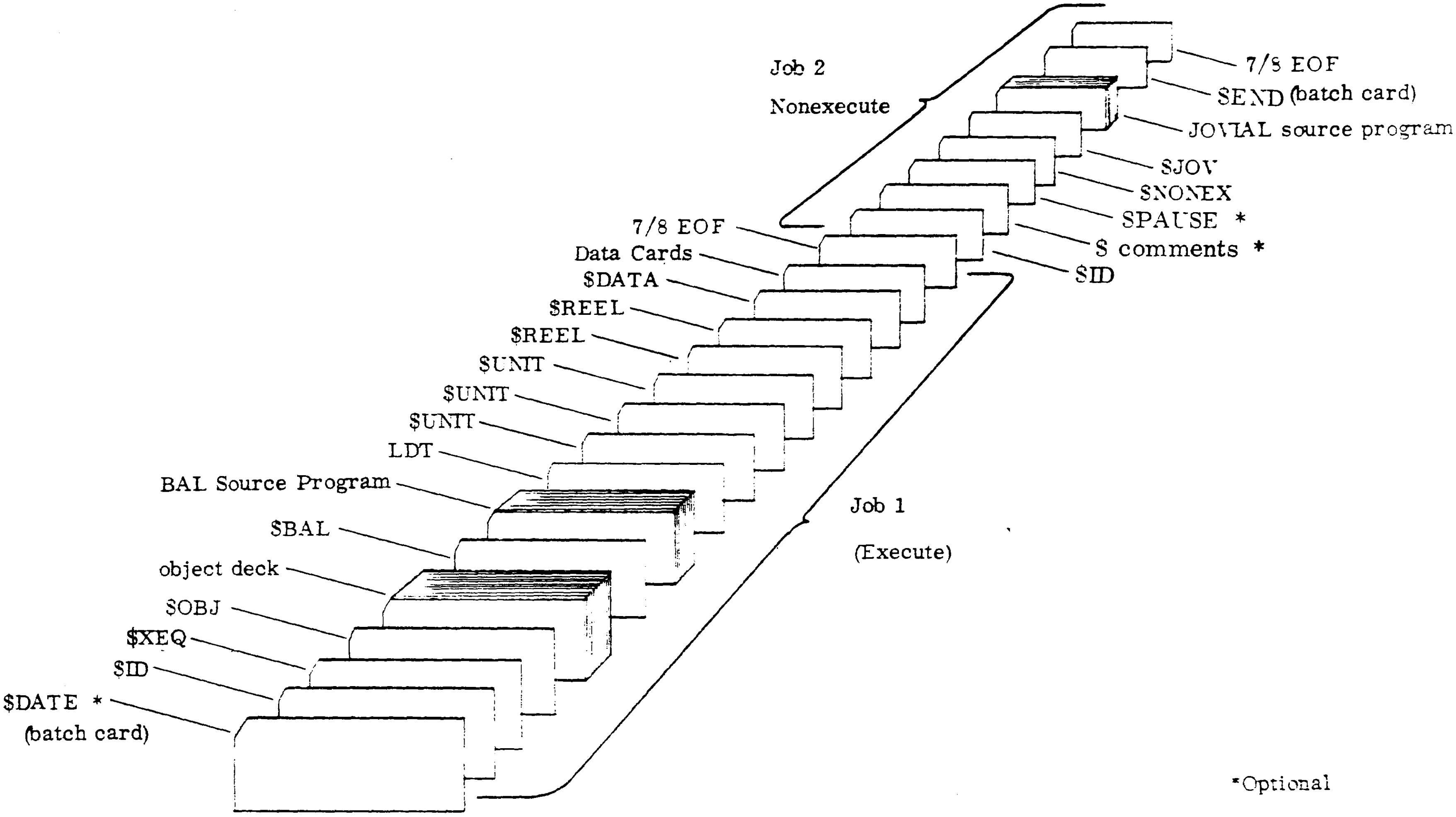
2.2.2.1 $ID Control Card

The $ID control card is required for proper identification of each job in a batch. There must be one $ID control card for each job. It precedes all other job control cards. The format of the $ID control card is:

Start Column	Content	Meaning
1	$ID	This is an identity card.
10	identity	This field contains six or less characters unique in the batch.
16	comment	This field can contain additional data, such as the programmer's name or building. This card is typed through the last non-blank column.

```
$ID          TEST 01 DOE, J. IBM BLDG 13

$            FIRST TEST OF A SERIES

$PAUSE       VERIFY TAPE MOUNTINGS

$ENV         SES=2, SYS=NS0621, SPT=PGM001, MOD-USER

$XEQ         NOSPT, NOLIB, NOCOMP

$NONEX       NOCOMP, NOLIB

$DATA

$REEL        30, P, 1, 2, 3

$UNIT        TAPE 43, USERIO, 04C, )08

$UNIT        TAPE 35, IOPACK, FXB, RL00810, BF10, BUF, 7TRACK, D800, BCD, EVEN

$UNIT        TAPE 27, IOPACK, VAR, 9TRACK

$UNIT        READ 28, IOPACK

$AUX         NOSPT
```

Figure 2–1. Sample Control Cards



*Optional

Figure 2–2. Sample Deck for 2-Job Batch

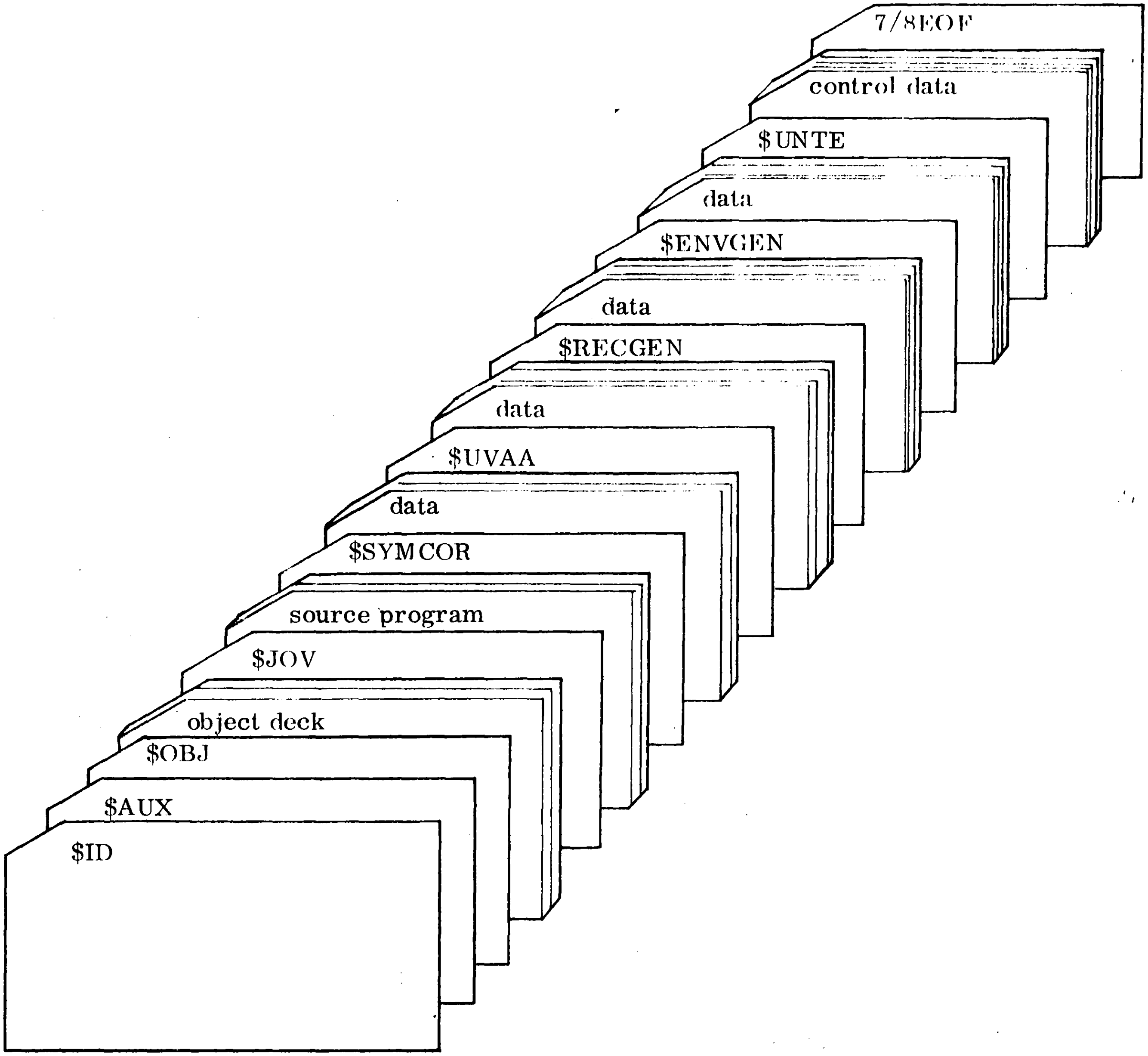


Figure 2–3. Sample Job Deck for Generating NAS Master Tape

2.2.2.2 $PAUSE Control Card

The $PAUSE control card causes the monitor to enter a wait loop until the operator depresses the ENTER key on the console typewriter; $PAUSE may be used to provide operator instructions. This control card is optional; if used, it must precede another system control card. The format of the $PAUSE control card is:

Start Column	Content	Meaning
1	$PAUSE	This is a pause.
16	comments	This field contains any alphanumeric character but cannot exceed column 72 in length. Data from this field is printed on the system output unit and typed on the console typewriter before the Monitor enters the wait loop.

2.2.2.3 $(Comments) Control Card

The $(comments) control card contains information to be placed on the system output unit and typed on the console typewriter. This control card is optional; if used, it must precede another monitor control card. The information on this card does not affect the processing of the job. Several may be used if necessary. The format of the $ comments control card is:

Start Column	Content	Meaning
1	$	This is a comments card. Columns 2–15 inclusive must be blank.
16	comment	This field contains any alphanumeric information including embedded blanks. The field cannot exceed column 72.

2.2.2.4 $PRINT Control Card

The format of a $PRINT card is:

Start Column	Content	Meaning
1	$PRINT	This card is similar to the $(comment) card.
16	comment	This field contains any alphanumeric information including embedded blanks. The field cannot exceed column 72.

The difference between $(comment) and $PRINT is that the information in column 16–72 is placed on SYSOUT *only*. $PRINT and $(comment) cards may be mixed in any order.

2.2.2.5 $ENV Control Card

The $ENV control card causes the monitor to verify that the job environment fulfills specified requirements. This card is optional. If used, $ENV cards must follow the $ID cards and precede the job-type control card. When specifying several parameters, they are placed on one card separated by commas, no blanks; more than one $ENV card may be used. The format of the $ENV control card is:

Start Column	Content	Meaning
1	$ENV	This is an environment control card.
16	SES=a	At least "a" (some number) storage elements are required. Nine is the maximum value allowed.
	SYS=bbbbbb	The characters bbbbbb represent a valid system tape label indicating the utility system tape which must be used to process this job.

NOTE: In case of error, depress the ENTER key to terminate this job, and rerun job with this specified system tape (i.e., no "OPERATOR OVERRIDE" capability for SYS = bbbbbb).

	LIB=cccccc	The characters cccccc represent a valid library tape label specifying the library tape required for this job.
	SPT=eeeeee	The characters eeeeee represent a valid SPT tape label. This field has the same meaning and effect as the LIB field.
	CMP=dddddddd	The characters dddddddd represent a valid compool name. This field has the same meaning and effect as the LIB field except that the compool name is specified and not the compool tape label. A full eight columns must be allocated for the name, using trailing blanks if appropriate.
	CLB=fffffff	This option is to be used with MLC tapes. The characters fffffff represent a valid compool name. This field has the same meaning and effect as the .LIB field except that the compool name is specified and not the MLC tape label. A full eight columns must be allocated for the name, using trailing blanks if appropriate.

NOTE: In case of error, the monitor types "REQUIRE" plus this field from the $ENV card, unloads the erroneous tapes, and awaits operator action. One of three actions is expected: the operator (1) loads the requested tapes and types CONT to request a check of its label, (2) types SKIP to omit a job that cannot be executed properly, or (3) types OVER if he believes the job should be run with the tape he has mounted. In this last case, "OPERATOR OVERRIDE" will be printed with the program output.

Start Column	Content	Meaning
	WK1	.WORK1 is required.
	WK2	.WORK2 is required.
	WK3	.WORK3 is required.
	MOD=SYST	This job is run in system-debug mode.
	MOD=USER	The problem program requires USERIO facilities.

The $ENV card advises the monitor to demand certain conditions. The programmer is obligated to furnish the operator with enough information to meet those conditions, such as correct tape slot numbers and a clearly marked request form that agrees with the $ENV cards.

Any or all of the $ENV options may be used, in any order, on one or more $ENV cards.

Exception: The option CLB or CMP, if used, must be the last option on a card.

Recommendation: The options SES and SYS be first when used.

NOTE: Current design of unit assignment logic prohibits the explicit or implicit release of WORK2 via any combination of options input on $ENV or $XEQ/$NONEX control cards.

If a user desires the priority effects of the $ENV card without specifying required tape labels, he may substitute six (or eight) asterisks (******) for a library, compool, or SPT label to insist only on the right kind of tape being mounted, or six (or eight) dollar signs ($$$$$$) to insist on a drive being assigned with no label checking.

2.2.3 Job-Type Control Cards

One of three job-type control cards may follow the $ID control card for the job. These cards indicate to the monitor the nature of the jobs that follow them.

a. $XEQ – Problem program is to be executed.

b. $AUX – Object decks are to be placed on the system auxiliary tape but are not to be executed.

c. $NONEX – Problem program is not to be executed.

2.2.3.1 $XEQ Control Card

The $XEQ (execution) control card indicates that the job consists of programs that are to be executed immediately after processing. The format of the $XEQ control card is:

Start Column	Content	Meaning
1	$XEQ	This field indicates an execution call is being made.
16	NOSPT	This field indicates no symbolic program tape is used.
	NOLIB	This field indicates the library is not used.
	NOCOMP	This field indicates compool is not used.

Any combination of options is valid; however, options must be separated by commas and no embedded blanks are allowed.

It is erroneous to declare a unit not required on a job-type card after declaring it essential on a $ENV card. This or any other error in options causes the defective option field and any following fields to be ignored.

Errors that preclude further processing and/or execution in a subprogram normally do not stop the processing of other subprograms in a job but stop execution if further job processing cannot continue.

2.2.3.2 $AUX Control Card

The $AUX (auxiliary) control card indicates that system processors are to place generated object decks on the auxiliary tape (.AUXIL) to be read by another processor. A typical use of the $AUX control card is when BAL Assembler and adaptation assembler outputs are placed on .AUXIL for the NAS System Edit subprogram.

The format of the $AUX control card is:

Start Column	Content	Meaning
1	$AUX	This is a $AUX card.
16	NOSPT	Same as for $XEQ.
	NOLIB	Same as for $XEQ.
	NOCOMP	Same as for $XEQ.

2.2.3.3 $NONEX Control Card

The $NONEX control card indicates nonexecution object decks are not placed on .AUXIL, not executed, nor may the user call a processor which reads .AUXIL. The format for the $NONEX control card is:

Start Column	Content	Meaning
1	$NONEX	Do not attempt execution.
16	NOSPT	Same as for $XEQ.
	NOLIB	Same as for $XEQ.
	NOCOMP	Same as for $XEQ.

2.2.4 Processor Control Cards

If a $XEQ, $AUX, or $NONEX control card is used, it must be followed by one or more processor control cards, each of which may be followed by a source program or other deck according to individual processor requirements. These cards indicate to the monitor the processors which are to be called during the job. The control card and its associated processor are listed below.

Control Card	Processor
$BAL	BAL Assembler
$JOV	JOVIAL Compiler
$OBJ	Object Blocking Routine

Details on the content and format of these cards can be found in the relevant Program Reference Manual, with one exception: the $OBJ Card. The Object Blocking Routine has one function; the routine copies and "blocks" object decks on .AUXIL. The object decks are mostly TXT (text) cards, each of which contains up to 56 bytes of assembled data. The TXT cards are packed into records of up to 272 bytes.

No options are necessary on the $OBJ card; comments are permitted starting in column 16.

2.2.5 Input/Output Control Cards

The I/O requirements of system processors are met by "system units". A system unit is normally either a tape drive, a printer, a card reader/punch or a console typewriter. Other types of devices (i.e., "non-standard I/O") may become system defined units under certain conditions (see paragraph 2.2.5.1.3). The utility monitor determines the device to be assigned as a standard system unit and controls the operation. Tape system units are described in detail in paragraph 2.3.5.

A problem program may require its own input or output devices which must be defined. The monitor uses the information on input/output control cards to make unit assignments. As many of these cards as are needed may be used, and they must appear immediately after the last problem program deck for the job. Whenever possible, units are assigned to alternate channels in the order in which their control cards appear.

The following discussion applies to non-system units.

2.2.5.1 $UNIT Control Card

The $UNIT control card defines the I/O unit to be used. There must be one $UNIT control card for every tape or other unit used by the job. There are two forms of $UNIT card: (1) an IOPACK unit card for normal use, utilizing service routines supplied with the utility monitor and (2) a USERIO unit card for cases where a user desires services not available from IOPACK.

2.2.5.1.1 IOPACK $UNIT Card. The format of the IOPACK $UNIT control card is:

Start Column	Content	Meaning
1	$UNIT	This is a $UNIT card.
10	TAPE	I/O unit is a tape.
	READ	I/O unit is a card reader.
	PUNCH	I/O unit is a card punch.
	PRINT	I/O unit is a printer.
16	xx,	The character xx represents a two-digit logical tape number, 20–99 inclusive. This number is used in the problem program to refer to the unit.
19	IOPACK,	Monitor I/O routines are used.

NOTE: The preceding specifications are mandatory; the following fields are used only when tape is specified as the type of I/O unit to be used. Some of these fields are not mandatory for tape units and may be omitted. The selected fields must be in the following order with no embedded blanks. Commas do not enclose omitted fields.

Start Column	Content	Meaning
26	VAR	Unblocked, variable-length tape records are used.
	FIX	Unblocked, fixed-length tape records are used.
	VBL	Blocked, variable-length tape records are used.
	FXB	Blocked, fixed-length tape records are used.
	RLn	This field represents the physical record length in bytes. The character n represents a 5-digit number, with leading zeros if needed. This field is omitted if variable-length, unblocked, unbuffered records are specified. For fixed-length records, n indicates the number of bytes in a record. (This number need not be repeated in a write tape operation.) For variable-length, unblocked, buffered and variable-length, blocked records, n indicates the largest possible number of bytes in a record; this length determines buffer size. Maximum allowable physical record length is 65,535 bytes. Minimum buffer size is 16 bytes. Minimum logical record length is 16 bytes for VAR and FIX type records and 16 divided by BFn for FXB type records. No minimum length is established for VBL type records. Maximum logical record size is RLn for VAR and FIX type records and RLn divided by BFn for FXB type records. Maximum logical record size for VBL type records is RLn–4.
	BFn	This field specifies the blocking factor. The character n represents a 2-digit number specifying the number of logical records per physical record for FXB type records. This field can be omitted if the blocking factor is one or if the records are of variable length.
	BUF	The monitor establishes two buffers for the file. This field can be omitted if tape records are VAR; if so, an unbuffered read or write operation is assumed, An error results if this field is omitted for any other type of record.
	7TRACK	This field indicates a 7-track tape unit is used.
	9TRACK	This field indicates a 9-track tape unit is used.

Start Column	Content	Meaning
	D200 D556 D800	This field is required for a 7-track tape and indicates the density of the tape to be used.
	BCD	IOPACK uses the code-translator mode of operation. This option is invalid if VBL is specified.
	BIN	IOPACK uses the byte-converter mode of operation.
	6 BIT	IOPACK uses the 6-bit mode of operation. (For input, 6-bit characters are stored as 8-bit characters with two leading zeros. For Output, 8-bit characters are reduced to 6-bit characters by truncating the first two bits.)

NOTE 1: This field applies to 7 track tapes only. If the field is void and 7TRACK specified, BIN is assumed.

	EVEN	Even parity is used.
	ODD	Odd parity is used.

NOTE 2: This field applies to 7-track tapes only. EVEN may be used only with BCD or 6BIT mode of operation; if BIN is selected, odd parity is always assumed.

2.2.5.1.2 Problem Program USERIO $UNIT Card. The format of the USERIO $UNIT control card is:

Start Column	Content	Meaning
1	$UNIT	This is a $UNIT card.
16	xx,	The characters xx represent a two-digit logical unit number, 20–99 inclusive.
19	USERIO,	Monitor services are not used; the user supplies his own services.
26	yyy,	The characters yyy represent a three-digit hexadecimal device address.
30	zz	The characters zz represent a two-digit device type, 06–10 inclusive.

For each device type specified in a USERIO $UNIT card, the user must supply a device initiation routine and a device interrupt routine comparable to the routines supplied in MONIO.

If a user supplies both USERIO and IOPACK $UNIT cards for tapes, the USERIO $UNIT cards should appear first because their addressing is fixed (declared in the card) while any remaining tape units can be selected by IOPACK $UNIT cards.

Problem program $UNIT cards generally immediately precede a $DATA card (see paragraph 2.2.6).

Detailed instructions for writing USERIO service routines are provided in paragraph 2.10.

2.2.5.1.3 System Processor USERIO $UNIT Card. Some system processors may have specific need for use of "non-standard" I/O devices; e.g., TTY, FDEP, DISK. These devices may become system defined units (logical unit numbers 18–23) through use of a system processor USERIO $UNIT card. The format of this control card is nearly identical to the $UNIT card shown in paragraph 2.2.5.1.2, with the exception of the logical unit number values and the device address (column 26). Here, the device address may be MXX, SXX, or XXX, where M indicates any multiplexor channel (0,4,8), S indicates any selector channel (1,2,5,6,9,A) and X is a hex digit. This card must be placed immediately before the $ control card which calls the system processor, and defines a non-standard device to be used by that processor for a particular job.

This card is given special handling by processor LISTGN, which processes $UNIT cards. LISTGN determines whether the device requested is available by checking for I/O response on the indicated channel (or on all possible channels in the case of M – or S – type channel indicator) and device number. If the device is available, an entry for it is made in the system unit assignment table, a LISTIO is generated, and acceptance of the device is indicated on the console typewriter (1052).

The system processor must supply both a device initiation routine, and a device interrupt routine comparable to the routines supplied in MONIO.

At end-of-file on SYSIN, USERIO entries in the system unit assignment table are eliminated both logically and physically. Hence, a new USERIO $UNIT card for non-standard devices is required for each job ($ID.....7/8 EOF), but not for extra processor calls within a job.

2.2.5.2 $REEL Control Card

A $REEL control card is used for data tapes that must be saved, for multireel files, and when a file is to be built and saved.

Each $REEL control card relates to only one $UNIT control card. If the $REEL control card is omitted, the file is assumed to consist of one tape reel without file protection. In this case, the monitor will not pause to allow mounts unless (1) a system unit must be removed or (2) a prior job requested that the tape on the same physical device be saved. A save message is never issued when there is no $REEL control card for a unit.

If the programmer knows the reel number of the first or only tape to be used, he indicates this number on the $REEL control card. If the reel number is unknown or if any reel is satisfactory, the programmer punches NORL on the $REEL control card. The monitor then assigns a scratch tape to the file. (With respect to pauses for mounts, the monitor functions as indicated in 1 and 2 above.)

The format of the $REEL control card is:

Start Column	Content	Meaning
1	$REEL	This is a $REEL card.
16	xx,	The characters xx are the logical unit number from the corresponding $UNIT card.
19	N	The character N indicates file protection is not used.
	P	The character P indicates file protection is required.
21	xxxx	The characters xxxx represent the reel sequence number. The number is one to four characters; leading zeros may be omitted. The reel number is not an internal label which is checked by a program; it is an external label which must be checked by the operator. If the programmer does not wish to assign a reel
	NORL	number to an output tape, he should place NORL in the reel number field. This field must not be blank. A separate NORL must be used for each output tape not given a reel number.

The fields on the $REEL control card must appear in the order given, separated by commas.

NOTE: With the occurrence of each end-of-tape or end-of-file condition, control will be passed to the user specified end-of-file address. It is the user's responsibility to issue all save and mount messages after the initial ones supplied by the monitor.

2.2.6 $DATA Control Card

If data cards are used by the problem program in a job, the data deck must be preceded by the $DATA control card and must appear as the last physical portion of the job deck. The format of the $DATA control card is:

Start Column	Content	Meaning
1	$DATA	A data deck follows.

The $DATA card allows the programmer to use .SYSIN for input to his program and to include it in a job that might for example include assembly of the program that is to operate on the data.

2.3 TAPE USE AND FORMATS

2.3.1 Identification of System Units

System units and their prime functions are listed in Table 2–1. System units are all tapes except .INPUT, .PRINT and .PUNCH, which may be tape or card reader, printer, and card punch respectively. If tape I/O is used in place of card reader, printer, and card punch, tape units are sometimes called .SYSIN and .SYSOUT. This mode is referred to as "off-line" input or output. The tape unit .SYSOUT contains both print and punch images.

Table 2–1. System Input/Output Units

Symbolic Name of Unit	Unit Type	LU No.	Definition
.SYSTEM	Tape	1	System tape
.INPUT	*	2	System input
.PRINT	**	3	System print output
.PUNCH	**	4	System punch output
.AUXIL	Tape	5	Loader input, dump tape
.LIB1	Tape	6	Library tape
.COMP	Tape	7	JOVIAL Compool
.SPT1	Tape	8	Symbolic program tape
.WORK1	Tape	9	Generated source deck transmitted from JOVIAL compiler or Symbolic Corrector to BAL Assembler, secondary work tape for other processors as required.
.WORK2	Tape	10	Output tape for Compool Edit, System Edit, SPT Edit, and Library Edit primary work tape for other processors as required, including JOVIAL Compiler and BAL Assembler
.WORK3	Tape	11	Tertiary work tape for processors as required, including JOVIAL Compiler
.TYPE	Typewriter	12	System console typewriter

* – INPUT may be tape (SYSIN) or card reader

** – PRINT and PUNCH may be tape (SYSOUT) in which case print and punch images are produced on the same tape, or the printer and card punch respectively.

Common practice is to configure a 9020 into two sets: an "I/O" computer consisting of one CE, one SE, and four or five tapes and a "compute" computer consisting of one CE and the balance of the available SE's and tapes. With this configuration the I/O computer prepares .SYSIN's and prints .SYSOUT's, while the other computer performs the jobs on its current .SYSIN, using the faster speed of tape input and output. A net saving in time is realized because the subprogram running in the I/O computer, BATCH, is designated to keep several printers and card sets busy.

The monitor configures to itself as many tape drives as are available. If fewer than the twelve units shown in Table 2–1 are available, the monitor tries to continue if possible.

NOTE: Although there are 12 units in the list, the maximum number of tapes is 10; the typewriter is always on-line and .SYSOUT includes print and punch.

The basic, or minimum I/O complement is:

.SYSTEM

.TYPE

.INPUT

.PRINT

.PUNCH

.AUXIL

This configuration could exist with two tape drives: .SYSTEM and .AUXIL.

As more tapes are available, the monitor assigns them in the following priority sequence:

a. **.LIB1**

b. **.WORK2**

c. **.WORK1**

d. **.WORK3**

e. **.COMP**

f. **.SPT1**

Tape assignments can be altered by operator actions and by $ENV cards. Tapes requested by $ENV cards take precedence over the normal priority sequence shown previously.

NOTE: Current NOSS monitor unit assignment logic may cause switching of logical and physical units between jobs. Programmers are urged to include explicit mount and save messages within all programs. Message text should include both logical unit number and physical unit assigned. Current unit assignments may be obtained from the current assignment table. The communications region within the NOSS monitor points to the current assignment table.

Without explicit mount and save messages, the user may either save the wrong output tape or mount an input tape on the wrong physical unit if the operator is forced to rely on the unit assignments typed out at IPL time.

NOTE: Current design of unit assignment logic prohibits the explicit or implicit release of WORK2 via any combi nation of options input on $ENV or $XEQ/$NONEX control cards.

Section 1 deals with tape requirements for utility system processors.

The formats of the system unit tapes are in some cases preset and unchangeable. The system tapes which have preset formats are .SYSTM, .SYSIN and .SYSOUT. In other cases, notably for the work tapes and non-system units, tape format is determined by the program using the tape. For non-system units, tape formats are established by $UNIT cards; for system units, a supervisor call is provided to establish tape format.

2.3.2 Tape Format Terminology

Before explaining the types of record formats recognized by the utility monitor, it may be beneficial to define terms used in describing record formats. The following definitions apply to the Utility System.

a. Physical Record. A group of related fields of information treated as a unit, extending from interrecord gap (IRG) to interrecord gap is defined as a physical record.

b. Logical Record. More than one group of related fields of information can be blocked in one physical record. Each such group is defined as a logical record. The number of logical records within a physical record is the blocking factor.

NOTE: When reference is made to a "record" in this document, the reference is to a logical record. The user must become familiar with the manner in which logical records and physical records are accessed. Paragraph 2.5.2, IOPACK, gives a detailed explanation of this.

c. Unblocked Records. An unblocked record is one in which the physical record contains only one logical record.

d. Blocked records. A blocked record is one in which the physical record contains more than one logical record.

e. Fixed-length Records. Fixed length records are records which are all the same length. For example, if all the records on a tape were 24 bytes long, the records would be fixed length.

f. Variable-length records. Variable-length records are records which may vary in length.

2.3.3 Record Types

The utility monitor is equipped to handle four types of tape record formats as specified by $UNIT cards or by the SVC call to SYSDTF. Another record format, the off-line print format, may be used by the monitor, but there is no way for the programmer to specify this format. The formats are fixed-length, unblocked; variable-length, unblocked; fixed-length, blocked; variable-length, blocked; or off-line print. Each format is described below:

2.3.3.1 Fixed-Length, Unblocked Records (FIX)

The physical record is identical to the logical record. All physical records have the same length. All records appear exactly as they were placed on the tape. Figure 2–4 illustrates this format.

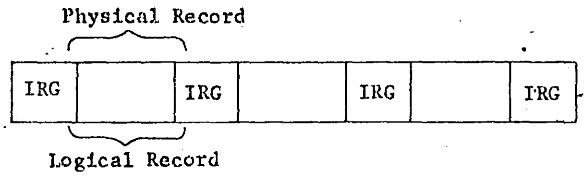


Figure 2–4. Fixed-Length, Unblocked Record Format

2.3.3.2 Variable-Length, Unblocked Records (VAR)

There is only one logical record per physical record, and physical records may vary in length. Figure 2–5 illustrates this format.

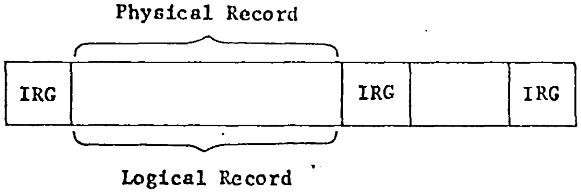


Figure 2–5. Variable Length, Unblocked Record Format

2.3.3.3 Fixed-Length, Blocked Records (FXB)

There is more than one logical record per physical record. The logical records are a fixed length, as are the physical records. Thus, the same number of logical records are in each physical record. The length of the physical record is equal to the length of a logical record, multiplied by the number of logical records per physical record (the blocking factor).

If padding is necessary to complete a physical record, it consists of 1 bits on 9-track tapes and 7-track binary tapes, and the character 9 (repeated as required) on 7-track BCD tapes. For 7-track, 6-bit padding consists of low-order 1 bits.

Figure 2–6 illustrates this type of record format.

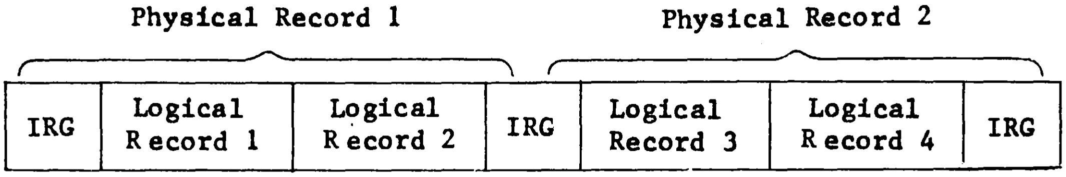


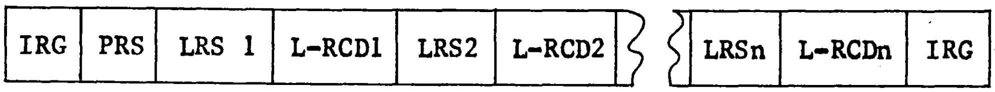
Figure 2–6. Fixed-Length, Blocked Record Format

2.3.3.4 Variable-Length, Blocked Record (VBL)

In these records, the length of the logical records may vary as well as the length of the physical records. The blocking factor for each physical record may also vary. When 7-track tapes are used, BCD and 6-bit mode are invalid for VBL records.

NOTE: Maximum physical record size for VBL is 32767 (i.e., X'7FFF'). SVC SYSDTF will not allow a buffer size greater than this maximum.

Figure 2–7 illustrates the format of a variable-length, blocked record.



Key:

PRS (physical record size) is a 2-byte field which indicates the number of bytes in the physical record, from inter-record gap to inter-record gap. The contents of PRS are equal to 2 (its own size) + 2 X (blocking factor) + the sum of the contents of all the LRS (logical record size) fields. The maximum physical record length permitted is 65,535 bytes.

LRS is a 2-byte field that indicates the size in bytes of the logical record immediately following. There is one LRS field preceding each logical record.

L–RCD is the logical record itself.

NOTE: PRS and LRS are generated by the monitor, and the programmer need not be concerned with these fields.

Figure 2–7. Variable-Length, Blocked Record Format

2.3.3.5 Off-Line Print Format Records (PRT)

This format is provided for use in off-line printing and punching. It is the format used in writing the .SYSOUT tape, and tapes written in this format can later be processed by the tape to print/punch programs provided for .SYSOUT tapes. However, the monitor makes no provision for reading PRT format records and any program needing to read this format must declare the format to be variable-length, unblocked and provide its own routine for unblocking the records.

The length of the logical records may vary as well as the length of the physical records. There is no fixed blocking factor. The maximum physical record size must not exceed 675 bytes if standard print/punch programs are to process the output tape. When 7-track tapes are used, 6BIT ODD mode is automatically selected and need not be specified.

The format of an off-line print format record is illustrated in Figure 2–8.

CC$_1$	L-RCD$_1$	RM	CC$_2$	L-RCD$_2$	RM		CC$_n$	L-RCD$_n$

Key:

a. CC (carriage control) is a carriage-control character or punch-image indicator as follows:

+: No space before printing (never a space after printing, so this means overprint)

blank One space before printing

0: Two spaces before printing

–: Three spaces before printing

1: Skip to 1 (restore) before printing

2: Skip to 2 before printing

3: Skip to 3 before printing

4: Skip to 4 before printing

5: Skip to 5 before printing

b. V L–RCD is an 80-character punch image

c. bcd X (Generated by the Monitor when writing 7-track tapes in PRT format; caller must use letter V in the image he supplies.) L–RCD is a 160-character punch image in which the first 80 characters represent the bit pattern of card rows 12–3 and the last 80 characters represent the bit patterns of card rows 4–9.

d. L–RCD is the logical record itself, in EBCD on 9-track tape, BCD for 7-track print records, and the format described above for 7-track punch records.

e. RM is a record mark character, hexadecimal (E0) on 9-track tape, hexadecimal (1A) on 7-track tape.

Figure 2–8. Off-Line Print Format Record (Sheet 1 of 2)

NOTE: Since all information necessary for blocking logical records is available to the user, it is permissible to block several records together with record marks and to present them to the monitor in a single call. However, this should not be done with punch images since the Monitor assumes that all punch images will be presented individually.

NOTE: Conversion from EBCD to BCD is performed in accordance with the following list:

EBCD	BCD
A to Z	A to Z
0 to 9	0 to 9
bl,., comma,*	bl,., comma,*
RM,–,/,$	RM,–,/,$
',@	'(quote)
(,%	(
),□	)
+,&	+
=,#	=
null (binary 0)	blank
all others	$

Figure 2–8. Off-Line Print Format Record (Sheet 2 of 2)

2.3.4 Buffers

The user must be aware of the buffer requirements for reading and writing tapes. There are two types of tape operations with which the user is concerned:

2.3.4.1 Unbuffered Operation

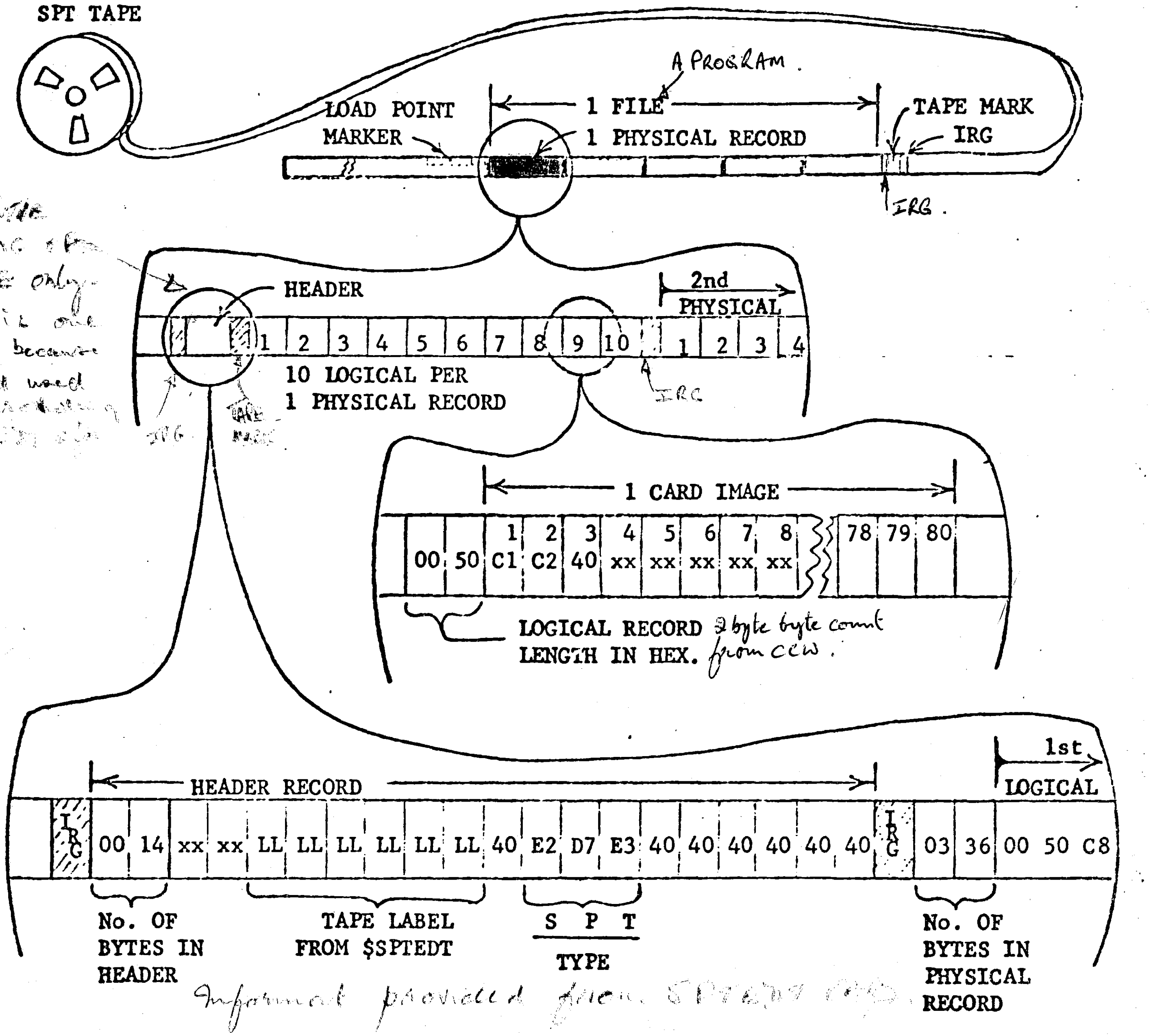
In an unbuffered operation the user need not specify a buffer. Data is directly read into or written from the user's data area. This operation is desirable when a program is cramped for space since tape records may be up to 65,535 bytes in length. However, if time is a more important factor than core storage, the double-buffered operation should be used.

2.3.4.2 Double-Buffered Operation

In a double-buffered operation, either the user specifies buffers (if the program is operating in system processor mode) or the monitor assigns buffer areas (if the program is a problem program). In either case two buffers are specified for input and for output. The user may specify, in the I/O call, an additional area in which to work on his data, or he may work on it in the buffer.

Double-buffered input allows faster processing time since the monitor keeps both buffers filled. On a read instruction, the data which has been previously placed in one input buffer is transferred to the User's input area while the next record is being read. On a write instruction, data from the user's output area is transferred to the output

SPT TAPE FORMAT

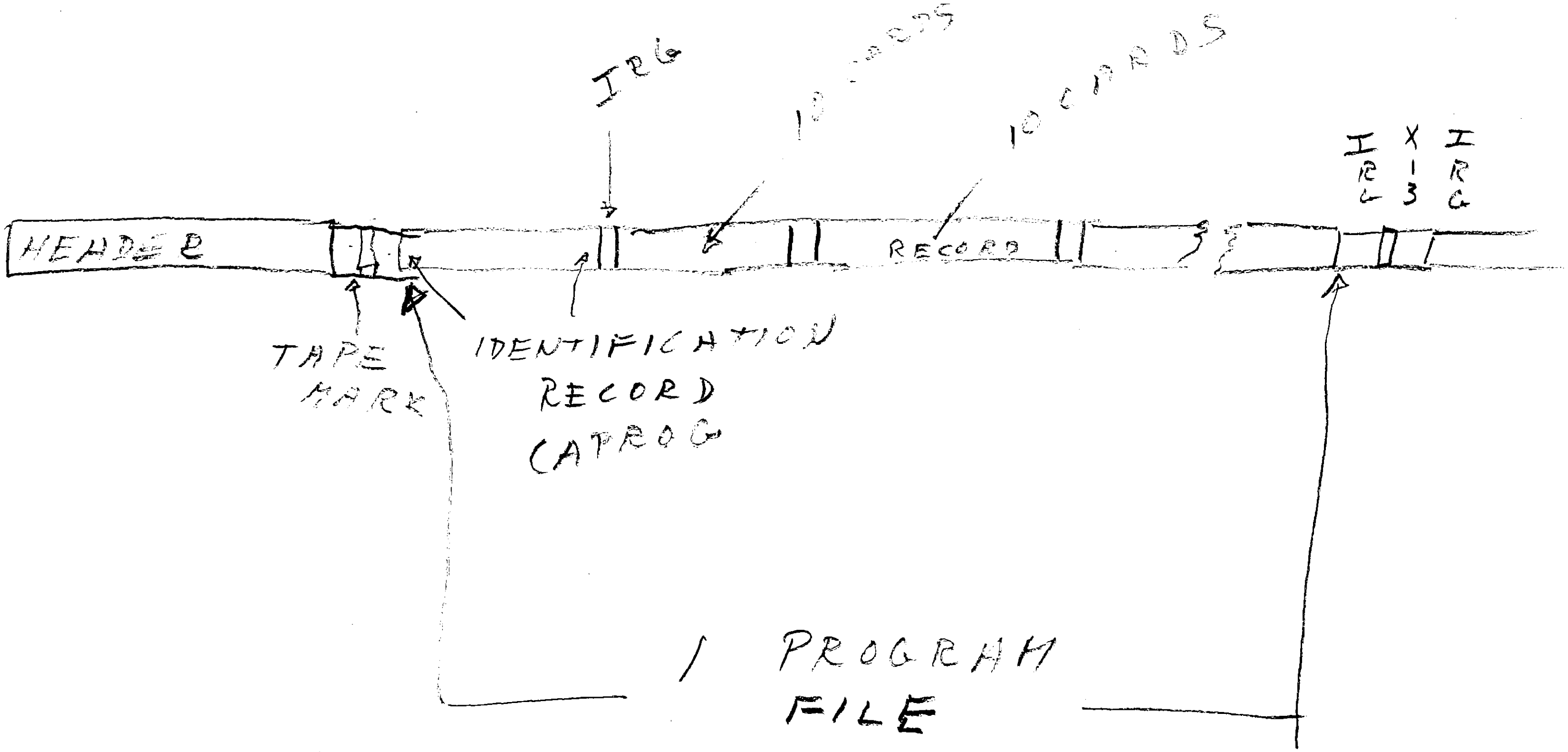


At the [illegible] of the TAPE only and it one FILE because [illegible] used for [illegible]

Informat provided from SPTEDT [illegible].

AJH 7/21/71

P. TO


HEADER
TAPE MARK
IDENTIFICATION RECORD CAPROG
IRG
10 CARDS
10 CARDS
RECORD
IRG
X-3
IRG
1 PROGRAM FILE

buffer; then control is transferred back to the user while data is being placed on tape. Paragraphs 2.5.2.2.4 and 2.5.2.2.5 give detailed descriptions of double buffered input/output.

2.3.5 System Unit Tape Formats

It may be of interest to the user to examine the formats of some system tapes (e.g., SPT tapes, Library tapes).

The user must understand that because a tape is mounted on a particular drive, it does not have to be in the same format as the tape usually associated with that drive. For example, any tape of any format may be mounted on the .SPT tape drive. When SPT tape format is referenced, the format is of the tape and not the tape drive.

The only concern of the user is that 7-track tapes must be mounted on 7-track tape drives and 9-track tapes must be mounted on 9-track tape drives.

Formats of the following tapes will be discussed in this section:

a. **.AUXIL tape**

b. **SPT tape**

c. **Library tape**

d. **Compool tape**

e. **MLC tape**

f. **SYSIN tape**

g. **SYSOUT tape**

If a header record for the tape exists, it is diagramed, and general format is given for the rest of the tape. For specific format information, the user should see the appropriate User's Manual or Program Logic Manual.

The format of the system tape need not be discussed since the user never accesses that tape.

2.3.5.1 .AUXIL Tape

The .AUXIL tape consists of variable-length, unblocked records. The records normally are 80 to 280 bytes in length. This is the format for the tape when records are placed there by object blocking, the BAL assembler, or one of the support processors.

The records are object deck card images. Each card image makes up one tape record except for TXT (text) cards which may be blocked up to 280 bytes.

The .AUXIL tape does not contain a header record.

2.3.5.2 SPT Tape

The format of an SPT tape is variable-length, blocked records. Each logical record is 80 bytes in length, and there may be a maximum of 10 logical records in one physical record. Therefore, the maximum length of a physical record on an SPT tape is 822 bytes.

Each 80-byte logical record is a card image. The header record consists of one record 20 bytes in length. Format of an SPT tape header record is shown in Figure 2–9.

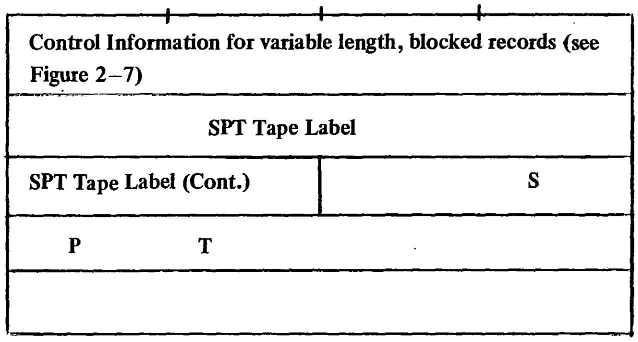

Control Information for variable length, blocked records (see Figure 2–7)
SPT Tape Label
SPT Tape Label (Cont.)
S
P
T

Figure 2–9. Format of an SPT Tape Header

2.3.5.3 Library Tape

The format of the Library tape is variable-length, unblocked records. The records may be a maximum of 20000 bytes in length.

The header record consists of one record, 16 bytes in length. Format of a Library tape header record is shown in Figure 2–10.

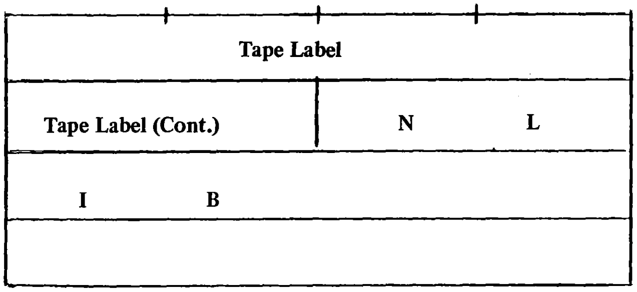


Figure 2–10. Format of a Library Tape Header

2.3.5.4 Compool Tape

The Compool tape format is of variable-length, unblocked records. The records may be 16 to 3584 bytes in length.

The header record (Compool Tape Index) is from 20 to 812 bytes in length, depending on the number of compools on the tape. Figure 2–11 shows the format of a compool tape header record.

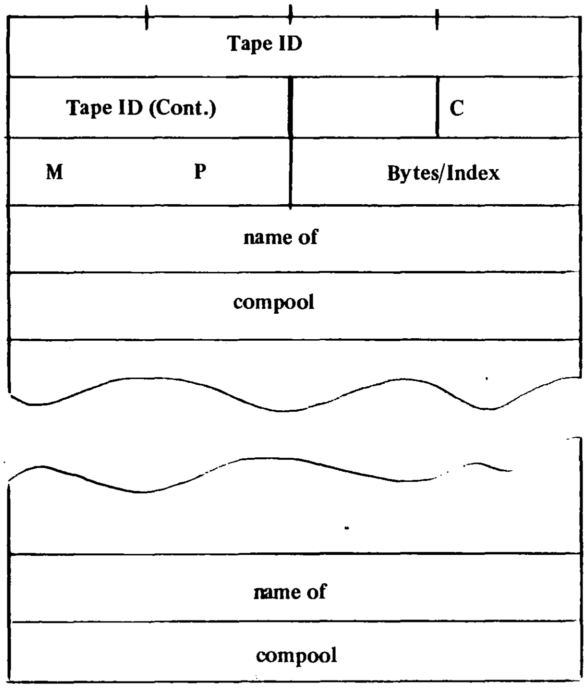


Figure 2–11. Format of Compool Tape Header

NOTE: There is no set relation between the order of compool names in the index and the order of compools on the compool tape. The index order is determined by the arrangement of .ADD control cards (preceding the .END control card) during an edit, whereas the order of compools on the tape is determined by the order of the input data decks. However, in an R-type edit, a newly added compool generally will be placed last in order in the index and first in order on the new tape.

2.3.5.5 MLC Tape

An MLC tape (combined Library-Compool tape) consists of variable-length, unblocked records. The first file contains the tape header; the second file contains the Library; and the third and subsequent files contain the compools. The records for the Library file are like those on the Library tape and the records for the compool files are like those on the compool tape.

An MLC tape contains a header which is 28 to 820 bytes in length. MLC tape header format is shown in Figure 2–12.

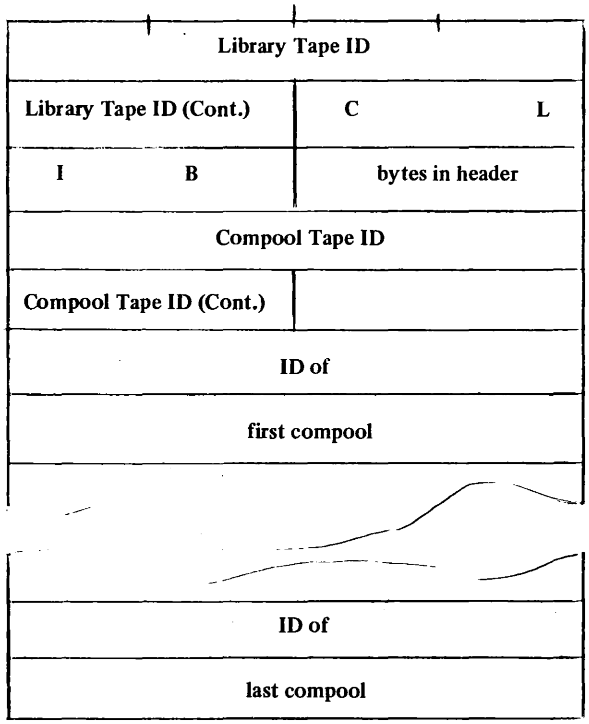


Figure 2–12. MLC Tape Header Format

2.3.5.6 SYSIN Tape

The utility system tape has fixed-length, blocked format. There are 810 bytes per record; each record consists of ten 81-byte card images. If there are less than 10 card images for the last record, the record is padded with X'F's to 810 bytes.

The SYSIN tape does not contain a header.

2.3.5.7 SYSOUT Tape

The SYSOUT tape consists of off-line print format records. The records contain output print images and may contain punched card images.

There is no header on a SYSOUT tape.

2.4 UTILITY SYSTEM TAPE AND STORAGE REQUIREMENTS

Each utility system processor requires a certain environment in which to operate. The following paragraphs identify the processor and specify corresponding tape and storage requirements.

The units .SYSTM and .AUXIL are required for every system processor. Therefore, these tapes are not included in the list of required tapes. The units .SYSOUT and .SYSIN are not included.

2.4.1 BAL Assembler – $BAL

The tape .WORK2 is optional. If it is present, and if necessary, the assembler uses it as a work tape. Whenever possible, it should be used in an assembly.

The storage required by the BAL assembler varies with the source program size and presence or absence of .WORK2. From 2 to 8 SEs may be required. For detailed limits see the BAL User's Manual.

Either .LIB or .COMP is necessary when assembling with a compool. In addition, .WORK1 is required for the PUNCHC option.

2.4.2 JOVIAL Compiler – $JOV

The .LIB unit is always required and must have either a Library or an MLC tape mounted on it.

The .COMP unit is required when compiling with a compool unless the .LIB drive contains an MLC with the correct compool. This unit is not needed when compiling without a compool.

The .WORK1 unit is always required, and the tape mounted there is used as a work tape. The final JOVIAL output is generated on this tape for input to the BAL assembler.

The .WORK2 unit is always required, and the tape mounted there is used as a work tape.

The .WORK3 unit may be released on a NONEX job, or on an XEQ job when the JOVIAL Compilation is the first deck in the job and the only compilation in the job. Otherwise, .WORK3 is required, and the tape mounted there is used as a work tape.

The storage required by the JOVIAL compiler depends on source program size and compool size. From 2 to 8 SEs may be required. See the JOVIAL User's Manual for detailed limits.

2.4.3 Compool Edit – $CMPEDT

The .WORK2 unit is always required and used as the unit for the output tape.

If the edit is an update of an old tape, one other tape drive is needed. The tape drive is usually .LIB unless that drive is not assigned. If not, .COMP drive is required.

The storage required by CMPEDT is variable, depending on Compool Source size. From 2 to 8 SEs may be required. See the CMPEDT User's Manual for detailed limits.

2.4.4 SPT Edit – $SPTEDT

The .WORK2 unit is always required and used as the unit for the output tape.

If the edit is an update of an old tape, one other tape drive is needed. The tape drive is usually .LIB unless that drive is not assigned. If not, the .SPT tape drive is required.

The SPT edit subprogram always operates in one SE.

2.4.5 System Edit – $SYSEDT

The .WORK2 unit is always required and used as the unit for output tape.

If the edit is an update of an old tape, .LIB is needed.

The SYSEDT subprogram always operates in one SE.

2.4.6 Library Edit – $LIBEDT

The .WORK2 unit is always required and used as the unit for the output tape.

If the edit is an update of an old tape, .LIB drive is needed.

The LIBEDT subprogram always operates in two SEs.

2.4.7 Object Blocking Routine – $OBJ

There are no tapes required for object blocking. Object blocking always operates in one SE.

2.4.8 Loader – $XEQ

The .LIB unit is required if any deck on .AUXIL contains a call to a Library routine.

The number of SEs required for the loader depends entirely on the size of the program(s) being loaded.

The loader subprogram itself operates in one SE.

2.5 FUNCTIONAL DESCRIPTION OF UTILITY MONITOR

The utility monitor initiates processing by calculating the size of the storage used, storing this value in the communications region, and protecting storage with two standard storage-protect keys: X‘0001’ for the monitor and X‘0010’ for the remainder of storage.

Next, the monitor reads from the console typewriter to signal that I/O unit reassignments may now be made. If standard unit assignments (those made by the monitor) are to be used, the operator responds with a TYPE message or depresses the ENTER key. If the operator gives a TYPE reply, all tape assignments will be typed out followed by the message SYSTEM WAITING. At this time the mounting of tapes may also take place. Depressing the ENTER key means that the standard units are ready and that processing may begin at once. The operator can select several messages (see Section 2.9) to alter the standard unit assignments. When changes are made in the standard unit assignments, the monitor issues messages similar to the following on the console typewriter, asking the operator to ready the new units:

```
**MOUNT
.INPUT     xxx
.PRINT     xxx
.PUNCH     xxx
.AUXIL     xxx
.WORK3     xxx
SYSTEM     WAITING
```

where xxx is the physical address of the unit.

The operator can then mount the proper tapes. (System processing does not start until the operator types in a CONT message.)

The monitor now begins reading input (from the system input unit) and searches for the $ID control card, which must be followed by a $ENV card or by one of three job-type control cards: $XEQ, $AUX, or $NONEX. If the monitor does not find one of these control cards immediately after the $ID card or any $ENV cards, it types a diagnostic message on the console typewriter, assumes that a $NONEX card is present, and continues processing the input file.

The $XEQ, $AUX, and $NONEX cards must be followed by control cards that indicate which components of the 9020 Utility System are needed. There is one control card for each processor. After examination of the control card, the monitor reads into storage the appropriate processor or processor phase from the system tape and transfers control to it. (A phase is a logical portion of a processor and constitutes one storage load.)

Any further loading of phases or processors is done through a supervisor call (SVC) from the processor to the monitor. The monitor reads in the next phase or processor and transfers control to it.

After the phase or processor has completed execution, the monitor ensures that all I/O requests are completed before the monitor initiates further processing. (Rewinds are permitted to overlap into the next activity.)

Uncorrectable errors encountered during system processing result in special action. The monitor issues a diagnostic message and reads forward to the next dollar ($) control card. If an I/O unit fails, the monitor asks the operator to select an option.

In a $XEQ job, when all object programs are ready to be loaded into storage and one or more $UNIT control cards are present, the monitor loads the LISTIO generator from the system tape. (LISTIO is described in paragraph 2.5.2.7.)

The generator then develops the object LISTIOs with their unit assignments, and the monitor types out:

```
**MOUNT IF REQUIRED
L.U. xx xxx xxxx
L.U. xx xxx xxxx
etc.
SYSTEM WAITING
```

where:

xx is the logical unit (LU) number.

xxx is the physical address of the unit.

xxx is either the reel number, NORL, or blank if a $REEL control card is not present. PROTECT may appear after a reel number if file protection was requested.

NOTE: If there is no $REEL control card, a SYSTEM WAITING message will be issued only if a system unit must be removed or a prior job requested a reel to be saved that is now on the same physical device indicated in the mount message.

After typing out the SYSTEM WAITING message, the monitor waits until the operator replies with a CONT message before initiating loading. When loading is finished, the monitor assigns buffer areas for each unit requiring them and then transfers control to the object program. This program has several functions available to it, which are performed through the monitor: changing the system mask, initializing the program interrupt entry point, and setting the protection keys.

If an I/O failure occurs during program execution, it will be handled by MONIO. If the failure is uncorrectable, the monitor will issue an appropriate message.

Looping can be terminated by the operator via the 1052 attention key or the console interrupt button. Termination of the problem program transfers control to the end-of-job portion of job control, which calls the MONIO mop-up routine. If the termination resulted from an error that prevented further processing or made processing meaningless, a terminal dump is taken and a SYSDUMP message is typed on the console typewriter. Subsequently, job control issues a JOB TERMINATED message.

After termination, the communications region is reset to initial conditions and the system LISTIOs are reinitialized. If a $REEL card was associated with any problem-program units, save messages are typed as follows:

****SAVE**
L.U. xx xxx xxxx
L.U. xx xxx xxxx
etc.

where:

xx is the logical unit (LU) number.

xxx is the physical address of the unit.

xxxx is either the reel number of NORL, whichever is specified.

There is no wait at this time unless standard system units were dismounted because of the previous run. In this case, the monitor reassigns units according to the information given on the $XEQ or $NONEX card and, as necessary, types this message:

****MOUNT**
.WORK1 xxx
(etc.)
SYSTEM WAITING

where:

xxx is the physical address of the unit.

The monitor then waits until the proper tapes are mounted, and the operator enters a CONT message before processing the next job.

The major sections of the monitor (job control, IOPACK, MONIO, and interrupt control) are described in detail in paragraphs 2.5.1 through 2.5.4.

2.5.1 Job Control

Job control performs three related functions: processor loading, control-card interpretation, and unit assignment. Since processor loading is entirely dependent upon control-card logic, these two functions are described together in the following text. Unit assignment is the selection of a physical device for each unit declared on a $UNIT control card. Involved with the selection is the generation of a LISTIO for each unit, based upon the information given by the $UNIT and $REEL control cards. System unit assignment is based upon a table that is a permanent part of the monitor (except in special cases. See paragraph 2.2.5.1.3).

2.5.1.1 Processor Loading and Control-Card Logic

Job control is responsible for interpreting the monitor control cards and messages and for acting upon them. The resulting actions, including processor loadings, are apparent from the card descriptions.

2.5.1.2 Input/Output Unit Assignments

To know at all times the status, assignment, and availability of I/O units, job control maintains two internal assignment tables and one availability table. One of the assignment tables reflects the current usage of system units for each job; the other is derived from the $UNIT control cards for each object program execution. The availability table keeps a record of the availability status of all I/O units attached to the machine, incorporating every change that occurs during the processing of a batch. These tables, together with the system and user LISTIOs, direct the I/O system in all its actions.

Input/Output units for problem program use are assigned to physical devices in the order in which the $UNIT cards are received. In addition, an attempt is made to assign each given unit to a channel different from the one used in the previous assignment. For example, if the first problem program unit was assigned to device 281, and, for the second assignment, two scratches – 180 and 280 – are available, 180 has the higher priority. However, if, for the second assignment, a scratch unit (280) and a LIB unit (182) are available, 280 is chosen, because reducing operator setup time is a more significant factor than the consideration of channel. This method of assigning problem program units optimizes the overlapping of I/O operations.

Job control assumes, at the start of each job, that the 12 system I/O units listed in Table 2–1 are available.

At the beginning of a batch, the .LIB1, .COMP, .SPT1, WORK1, .WORK2 and WORK3 tapes may be released from system assignment. This is done by means of RELEASE operator messages. (See the NOSS Operator's Manual.) Only tapes that will not be used by any job in the batch may be released. To free additional tapes, the USEREADER and USEPRINT/PUNCH operator messages will free the .INPUT and the .PRINT/.PUNCH tapes by allowing the on-line card reader, printer, and card punch to be the system I/O units. The units .SYSTM and .AUXIL are always required, must not be dismounted, and cannot be released, detached, or reassigned.

If the programmer requires use of a system tape unit, such as SPT, which might normally not be assigned, he can free some drives, for the current job only, by using the three options provided on the $XEQ, $AUX, and $NONEX control cards. This release has no effect upon other jobs in the same batch.

At the start of the $XEQ job, job control assumes that all system tapes are mounted, but it makes all assignments indicated by operator messages and by $XEQ options. Just before execution, it assigns to the drives the unique but arbitrary numbers specified by the $UNIT control cards in the problem program.

Having extracted the information from the $UNIT control cards, assigned the requested problem-program I/O unit numbers to physical tape drives, and allocated buffers accordingly, job control types out appropriate instructions to the operator. It then waits until the tapes are mounted and the operator signals that processing may continue. At this point, job control transfers to the first executable instruction in the problem program.

If an I/O unit is unavailable at the start of the batch, the operator informs job control by means of a DETACH message. Job control then tries reassignment if the detached unit contains a system component. Reassignment is performed in accordance with a priority list so that the most essential system units are assigned first. The monitor looks for a tape drive that does not contain a system component. If this fails, it tries to reassign a system unit with lower priority. After attempting reassignment of all detached system units, it finally checks for the presence of the basic system units before processing continues. The batch is terminated if any basic unit remains detached.

2.5.2 IOPACK

IOPACK is a set of I/O routines which are part of the Utility Monitor. Whenever a library routine or SVC call is initiated to perform an I/O function, the IOPACK routines actually execute the I/O commands. Both system processors and object programs use the IOPACK routines by using Library Routines or SVC calls.

IOPACK operates in Monitor state and uses a set of special I/O routines (MONIO) to execute its commands. IOPACK and MONIO communicate through parameter blocks called LISTIO's. These blocks contain information about the I/O units being used; there is one LISTIO for each I/O unit in use during the processing of a job. (LISTIO format is described in Section 6). IOPACK supports the following devices only: tape, card reader, card punch, printer, and console typewriter. The read-backward feature for tapes is not supported.

2.5.2.1 IOPACK Functions

The IOPACK routines perform general I/O functions for the programmer. The routines open and close files to be used by the job, transmit logical records to or from a buffer, determine whether a buffer is ready to be filled or emptied, and transfer to an end-of-file return after the last logical record has been taken from the buffer. IOPACK waits until a device is free before calling MONIO, thus ensuring that only one I/O request is ever pending for a single device.

2.5.2.2 Buffer Handling

IOPACK provides for 3 types of buffered input and output:

1. Unbuffered

2. Single-buffered

3. Double-buffered

Table 2–2 shows the requirements of I/O units vs. buffering.

2.5.2.2.1 Unbuffered I/O Operation. In an unbuffered operation, IOPACK does nothing until it receives a request to transfer data. Then it requests that MONIO transfer one physical record to or from a user specified area. It waits for the operation to be completed and returns control to the user.

Table 2–2. I/O Units vs. Buffering

	Buffering		
I/O Unit	Unbuffered	Single Buffered	Double Buffered
Tape	O		O
Card Reader		Input	

Table 2–2. I/O Units vs. Buffering (Continued)

	Buffering		
I/O Unit	Unbuffered	Single Buffered	Double Buffered
Card Punch		Output	
Printer			Output
Console Typewriter	Input	Output	

where:

O = optional for input or output

Input = mandatory for input

Output = mandatory for output

Unbuffered operation is mandatory with console typewriter input and optional with tapes.

2.5.2.2.2 Single-Buffered Input Operation. In single-buffered input, IOPACK checks whether it has previously requested data from the device and, if not, requests that MONIO transfer one physical record to a monitor-assigned buffer area. It then waits for the operation to be completed and error-checked. Finally, IOPACK moves the data to the user-specified area, requests the next record from MONIO, and returns control to the user. Single-buffered input is mandatory with the card reader but not used elsewhere.

2.5.2.2.3 Single-Buffered Output Operation. In single-buffered output, IOPACK checks for successful completion of any previously requested operation and then moves the current data from the user-specified area to a monitor-assigned buffer area. It then requests that MONIO output the record and returns control to the user. Single-buffered output is mandatory with the on-line punch and console typewriter but not used elsewhere.

2.5.2.2.4 Double-Buffered Input Operation. In double-buffered input, IOPACK first checks whether it is unblocking blocked records. If so, and if the last logical record delivered was not the last record in a block, all references to MONIO can be skipped because the next logical record is known to be available. Otherwise, IOPACK checks whether it has previously "opened" the device and, if not, requests that MONIO transfer one physical record to one of two monitor-assigned buffer areas. IOPACK then waits for the operation to be completed and error-checked. In case of end of file or catastrophic error, it "closes" the file and makes an appropriate exit. For success, it requests that MONIO read the next physical record to the alternate monitor-assigned buffer area. After MONIO operation or without it, as appropriate, IOPACK determines the size of the next logical record and either moves it to the user-specified area or informs him via general register of its location in the buffer area. Double-buffered input is optional with tapes.

2.5.2.2.5 Double-Buffered Output Operation. In double-buffered tape output, IOPACK first checks whether it is blocking variable-length logical records into multirecord physical blocks (VBL). If so, and if the current logical record will not fit in the block being constructed, IOPACK checks for successful completion of any previously requested operation, requests that MONIO output the record, and marks the alternate buffer as current and empty. In either case, for VBL operation, IOPACK finishes by moving the current data from the user-specified area to the next available location in the current buffer area. For all other record formats (FIX, VAR, and FXB), IOPACK moves the current data from the user-specified area to the next available location in the current buffer area. Then it checks whether the buffer is full (always true for FIX and VAR) and, if not, returns control to the user. If the buffer is full, IOPACK checks for successful completion of any previously requested operation, requests that MONIO output the record, and marks the alternate buffer as being current and empty before returning control to the user. Double-buffered output is optional with tapes.

In double-buffered output to an on-line printer, IOPACK checks whether the next buffer to be used is empty. If not, it waits with interrupts enabled until an interrupt occurs and MONIO marks the buffer empty. As soon as an empty buffer is available, IOPACK moves in the print image. If the opposite buffer is now empty, IOPACK requests that MONIO initiate printer operation (not necessary if the printer was currently busy).

In closing an output file (done only in response to a request to write a tapemark), IOPACK ensures that all logical records in the buffers are successfully written before actually writing the tapemark.

In processing multireel files, IOPACK provides for unloading reels when an end-of-file or end-of-tape condition is detected, for necessary operator communication, and for opening the next reel with no indication to the using program. Only on the last reel specified will end-of-file or end-of-tape status be passed to the user.

2.5.2.3 IOPACK Record Formats

Files processed by IOPACK may consist of four record formats, as specified on the $UNIT control card, plus off-line print format (PRT). The records may be fixed-length, unblocked (FIX); variable-length, unblocked (VAR); fixed-length, blocked (FXB); variable-length, blocked (VBL) or off-line print (PRT). These formats are described in paragraph 2.3.2.

2.5.2.4 IOPACK Routines

IOPACK consists of 18 routines that perform I/O functions for processors and object programs. Table 2–8 lists and describes the IOPACK routines. Paragraph 2.7 describes the calling sequences for the various routines.

Table 2–3. IOPACK Routines

Class	Description (from or to)	How Called	Symbol	Page
Input	General Input (any unit)	SVC	SYSRDS	2–34
	System Input Unit	BALR	SYSJN	2–34
	Tape Input	BALR	SYSRT	2–35
	On-Line Card Reader	BALR	SYSRC	2–35
	Console Typewriter Input	SVC	SYSTRE	2–36
Output General Data	General Output	SVC	SYSWRS	2–36
	Tape Output	BALR	SYSWT	2–36
Output Print Images	General Print	SVC	SYSPRS	2–37
	System Output Unit Print	BALR	SYSJT	2–38
	On-Line Printer	BALR	SYSPRT	2–38
	Console Typewriter Output	SVC	SYSTWR	2–38
Output Punch Images	General Punch	SVC	SYSPUN	2–39
	System Output Unit Punch	BALR	SYSJH	2–39
	On-Line Punch	BALR	SYSPC	2–39
Miscellaneous	Set Return	BALR	SYSSR	2–41
	Set Return	SVC	SYSSTR	2–39
	Unit Control	BALR	SYSCT	2–41
	Unit Control	SVC	SYSCTL	2–39

2.5.2.5 IOPACK and System Input

The system input unit (.INPUT) contains input to the 9020 Utility System except for special messages entered at the console typewriter. The information on the system input unit is read into storage by the IOPACK job-in routine, and may be on punched cards or on an externally created tape.

2.5.2.5.1 Card Input. Card input to the utility system is in the form of the standard 80-column cards punched in Extended Binary-Coded Decimal Interchange Code (EBCDIC). The card is punched with one character per column, and each card column is represented internally in 8-bit code, equivalent to one byte of storage.

2.5.2.5.2 Tape Input. Each physical record on the system input tape consists of the image of 10 cards and contains 810 bytes. No heading precedes the first byte of the first card in the record, but separation is made between the bytes of different cards. The final byte of the 810 bytes is padding and consists of 1 bits. The 7/8EOF card is represented by a tapemark.

2.5.2.6 IOPACK and System Output

The system output unit (.OUTPUT or .SYSOUT) receives normal output from the 9020 Utility System except for special messages outputted to the console typewriter. The normal data are outputted by several IOPACK routines oriented to print and punch operations and may be directed to an on-line printer and punch or to a tape for a later tape to print/punch operation.

2.5.2.6.1 On-Line Output. Print output is in the form of EBCDIC images of 1 to 132 characters each via a 1403 Printer assumed to be equipped with a PN–1 (PL/1 character set) print chain.

Card output is in the form of 80-column cards punched in EBCDIC code.

2.5.2.6.2 Tape Output. Tape output is in PRT format with block size limited to 676 characters. Each block may contain both print and punch logical records.

2.5.2.7 LISTIO Tables

The LISTIO's are parameter blocks of data and I/O unit information that reside in a protected area of storage. There is one LISTIO for each I/O unit used by the programs within a specific job. The contents of each problem program LISTIO are generated at the beginning of the job from $UNIT and $REEL control cards and are modified by IOPACK routines to provide MONIO with the necessary information. System LISTIO's are generated by the Monitor at the beginning of each batch (and/or when a system processor USERIO $UNIT card is validated. The problem program and system LISTIO's contain all the information needed for communication between IOPACK and MONIO. Figure 2–13 shows LISTIO format.

2.5.3 MONIO

MONIO consists of machine I/O and interrupt routines that maintain asynchronous operations on all I/O units that have received I/O requests through IOPACK. MONIO executes the requests, services I/O interrupts, handles error recovery and typing, and performs I/O closing (mop-up) after each system phase and each job. MONIO consists of three major routines, each of which is executed in the monitor state. MONIO communicates with IOPACK through LISTIO's, which are blocks of data that contain information regarding every I/O unit being used by the job. The communication is accomplished by flags, which are set by MONIO to indicate the current status of the I/O unit. For example, the activity-requested flag is turned on whenever an I/O operation is awaiting initiation. Another flag that is used frequently by MONIO is the unit-free flag. This flag is turned off whenever an operation is initiated on the associated I/O unit, and is turned on when the operation is completed and the unit is again available for use. The routines in MONIO use these and other flags in the LISTIO to coordinate their functions.

2.5.3.1 Input/Output Interrupt Routine

The I/O interrupt routine is initiated whenever an I/O interrupt occurs. The old I/O PSW is placed in a PSW stack, and a new PSW is fetched. The routine then searches the LISTIO's to find the cause of the interrupt. MONIO tests for errors and determines whether they are correctable, sets the corresponding error flags, and queues the appropriate error correction procedure in LISTIO. MONIO then sets a pointer for the select-scan routine to select a unit on the same channel as the interrupt and transfers control to the select-scan routine.

2.5.3.2 Select-Scan Routine

The select-scan routine can be entered from IOPACK or from the MONIO interrupt routine. IOPACK calls the

BIT

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 23 24 31 32 33 34 35 36 37 38 39 40 45 46 55 56

Byte Displacement Hexadecimal	Byte Displacement Decimal								
0	0	Pointer to Next LISTIO on This Channel				Pointer to Buffer 0			
8	8	Pointer to Buffer 1				Current Record Pointer			
10	16	Unused				Physical Record Length for Buffer 0			
18	24	Physical Record Length for Buffer 1				Current Logical Record Length			
20	32	Channel Command Word							
		Command Code	Data Address			Flags	000	00000000	Count
28	40	Channel Address Word				Pointer to Reel Identification Table			
		Key	0000	Pointer to Channel Command Word					
30	48	IOPACK Flags: FIRST FREE EOT REQU USER BUF ERROR 0 or 1 LEVEL 2 BCC 0 BCC 1 CHAIN PT/PCH ERROT BASIC CHAIN		Buffer Size		Blocking Factor	Current Record Number	Number of Reels	Current Reel Number
38	56	Device Type	Record Type	Tape Mode	Operation Requested by IOPACK	MONIO Flags: COR REQU FREE NOT READY PCH REPE BAKSPC REQU TIE ERASE	Track in Error Information	Device Address	
40	64	File Count		Block Count		MONIO Correction Operation	Tries Count	Backspace Count	
48	72	End-of-File Return Address				Error Return Address			

Figure 2–13. LISTIO Format

select-scan routine whenever an I/O request is to be initiated; the MONIO interrupt routine calls the select-scan routine to initiate error correction or stacked operations.

The select-scan routine searches the LISTIO's associated with the specified channel to determine whether any I/O requests, including error correction operations, are pending. If a request is found, MONIO attempts to initiate the requested operation on that channel. If the request is initiated, the device-free and activity-requested flag in the LISTIO are turned off. If the request cannot be initiated because the channel is already in operation, MONIO returns control to the point of interruption; if the request cannot be initiated because the unit is not ready, a message is typed on the console typewriter and the not-ready flag is turned on in the LISTIO.

2.5.3.3 Input/Output Mop-Up Routine

The I/O mop-up (closing) routine is entered after each system phase as well as after each job. If the processing has ended successfully, the routine scans the LISTIO's to ensure that all requests have been completed before the monitor proceeds to the next phase or job. If a unit-free flag is off, MONIO pauses until the flag is turned on; if the unit-free flag is on, MONIO moves to the next LISTIO. When an activity-requested flag is on, MONIO waits until the flag is turned off and the corresponding unit-free flag is turned on. MONIO does not wait for the completion of rewind operations.

If the mop-up (closing) routine is entered from a phase or job that ended unsuccessfully, the unit-free flags are turned on for all LISTIO's. Any stacked requests are eliminated, and any operations in process are ignored.

2.5.3.4 MONIO and Error Correction

When an interrupt is caused by an I/O error, MONIO attempts to correct the error. When an error correction fails, messages from MONIO that describe the error are typed on the console typewriter.

2.5.3.5 Not-Ready Conditions

If a unit is not ready when requested, MONIO issues a message on the console typewriter, requesting the operator to ready the unit. Operator response to the message is not expected. After the unit is made ready the program retries the original request.

2.5.3.6 Initiation Errors

A channel has an initiation error if the condition code is 3 (channel not operational), if the condition code is 0 to a control command (channel has no status to report), or if any command is rejected. When an initiation error is encountered, MONIO issues a message on the console typewriter, and the operator is expected to respond.

2.5.3.7 Tape-Read Errors

If a tape-read error isencountered, MONIOinitiates an error correction procedure. This includes backspacing and rereading the record five times. If the error persists, the sense bytes are typed out, and the operator is expected to respond.

2.5.3.8 Tape-Write Errors

If a tape-write error is encountered, MONIO makes several attempts to rewrite the record (including two skip and erase commands). If the error persists, the sense bytes are typed out, and the operator is expected to respond.

2.5.3.9 Card-Read Error

When a card-read error is encountered, MONIO issues a message on the console typewriter. The operator is expected to run the cards out (no processing) from the reader, place these cards (usually three, including error card) in the read hopper, depress the START button of the card reader, and type REPE.

2.5.3.10 Card-Punch Errors

If a card-punch error is detected, MONIO issues a message on the console typewriter. The operator is expected to run the cards out (no processing) from the punch, discard the error card, place the blank cards in the punch hopper, and type REPE. Usually, a faulty card will fall into the selected hopper and should be discarded too.

2.5.3.11 Print Errors

If a print error is detected, MONIO issues a message on the console typewriter, and the operator is expected to respond. If the operator types REPE, the print operation will be retried.

2.5.4 Interrupt Control

Interrupt control is designed to handle any interrupts that may occur during processing. There are five interrupt-control routines, each dealing with one of the five major types of interrupts: program, input/output, monitor call, external, and machine.

2.5.4.1 Program-Interrupt Routines

A program interrupt is caused by exceptions resulting from abnormal or improper use of an instruction or data. When this type of interrupt occurs, the current instruction is completed, terminated, or suppressed, depending on the specific interrupt.

There are 15 types of program interrupts: operation code, privileged operation, execute, storage protection, addressing, specification, data, point overflow, exponent underflow, significance, and floating-point divide interrupts. These interrupts are machine functions and are described in detail in *IBM 9020 Data Processing System – Principles of Operation,* Form A22–6852. In addition, execution of a supervisor call not recognized by the utility monitor is treated as a 16th psuedo-program interrupt to permit the user to specify action to be taken via SVC SYSPIN.

The cause of the interrupt is identified by interrupt code bits 28–31 of the old PSW. The instruction-length code in the old PSW indicates the length, in halfwords, of the preceding instruction. In cases where the instruction length is not available, the instruction-length field is zero.

Only four of the 15 types of program interrupts can be masked, as indicated in Table 2–4. An interrupt that can be masked causes a program interruption only if the corresponding mask bit (bit 36, 37, 38, or 39 of the PSW) is set to 1. If the corresponding mask bit is 0, the interrupt is ignored.

The Set Program Mask (SPM) machine instruction is used to mask or unmask the four program interrupts. This instruction is described in *IBM 9020 Data Processing System – Principles of Operation,* Form A22–6852.

At the start of execution of each job, the monitor initializes a set of address constants to indicate that each type of program interrupt will result in job termination, excluding the PSW markable program interrupts. These four special cases are set to ignore status (i.e., PSW bits set to zéro). At any time during execution, the user can specify or respecify use of his own recovery routines for individual interrupts and/or specify job termination for individual interrupts via the following calling sequences:

```
     CNOP      2,4
     SVC       SYSPIN
     DC        A(list)
```

where list is the address of a user array, consisting of a binary mask and one address constant for each routine affected. Each address constant represents the address of the programmer's corresponding interrupt servicing routine. If the programmer wishes the monitor to process the interrupt, he can use DC A(0) as the address constant.

The user array may be coded as:

```
     CNOP      2,4
LIST DC        X'xxx'
     DC        A(routine–1)
     DC        A(routine–2)
```

Table 2–4. Masking Program Interrupts

Type of Program Interruption	Interruption Code	Program Mask Bit	Action If Mask Bit Is Set to Zero
Fixed-Point Overflow	8	36	Information outside register is ignored.
Decimal Overflow	10	37	The overflow information is ignored.
Exponent Underflow	13	38	The result is made a true zero.
Significance	14	39	The result is made a true zero.

```
          .

          .

          .

     DC            A(routine–n)
```

where:

xxx is a 4-character, hexadecimal mask comprising 16 bits. Each of these bits (starting at the left) is associated with one of the 15 program interrupts (bit 16 being associated with unrecognizable SVC's) as ordered in *IBM 9020 Data Processing System – Principles of Operation,* Form A22–6852. For each bit that is set to 1 in the mask, an address constant, such as A (routine–1), must be furnished in the array. For example, a mask of:

0010010010011110

would mean that address constants are given for program interrupts 3, 6, 9, 12, 13, 14, and 15. This would be coded as:

DC X′ 249E′

routine–1, routine–2, and routine–n are the symbolic names of the programmer's routines.

If the programmer has elected to service an interrupt himself and has initialized interrupt control as previously described, then upon occurrence of any such program interrupt four events will occur:

a. Interrupt control of the monitor will process the interrupt initially and determine the correct address (specified by the SYSPIN monitor call) to which control should be passed.

b. At the time interrupt control transfers to the programmer's routine, the contents of the general registers and the floating-point registers will be the same as at the time of the interrupt. The PSW will be unchanged except for its instruction counter field and Storage Protect Key. The new PSW Storage Protect Key will be matched to the programmer's interrupt routine unless that routine has a key of zero. If the routine has a key of zero, the new PSW Key will be the same as that in the old PSW prior to the interrupt.

c. After executing any corrective action required, the routine should restore the general registers as they were at the time of entry and then exit through the monitor. Either of two exits may be used:

SVC SYSRET or SVC SYSRTA

DC AL4 (address)

d. The monitor interprets the SYSRET exit as requiring return to the point of the program interrupt. The SYSRTA exit may be used to cause return to a different address, as specified in the DC statement.

2.5.4.2 Input/Output Interrupt Routine

The I/O interrupt routine allows the monitor to respond to signals from I/O units. The programmer who uses the 9020 Utility System need not be concerned with how an I/O interrupt is processed, since the monitor has this responsibility. However, a brief explanation is included in the following text.

A request for an I/O interrupt can occur at any time, and more than one may occur at the same time. The requests are preserved in the I/O section of the computer until they can be honored. The requests are processed one at a time. An I/O interrupt can occur only if the current instruction has been completed and if the channel presenting the request is unmasked. The channels are masked by the system mask bits (0–6 and 16–19) of the PSW; masked I/O interrupts remain pending.

The interrupt code in the old PSW identifies the channel and I/O unit causing the interrupt (bits 20–23 and 24–31 of the old PSW). An interrupt routine is called in by the monitor to suit the specific type of I/O interrupt. The monitor finds the address of the routine, using the unit (device) field in the old PSW and information in internal tables, and passes control to the routine. When the interrupt routine has accomplished its function, control is returned to the point of the interrupt.

2.5.4.3 Monitor-Call Interrupt Routine

This interrupt occurs as the result of the execution of an SVC instruction. A major function of this interrupt is to switch the system status from the problem to the supervisor state, but the interrupt may also be used for other types of status switching. The interrupt code is used to instruct the monitor as to the particular function to be performed. The monitor calls available to the problem-programmer are listed in Table 2–5. These calls are explained in paragraph 2.8.

Table 2–5. Monitor Calls

Mnemonic	Decimal Code	Hex.	Class*	Description	Page
SYSPIN	20	14	P	Initialize program interrupts.	2–28
SYSCOM	21	15	N	Special monitor communication.	2–41
SYSTIM	22	16	P	Set up interval timer routine.	2–32
SYSIO	23	17	U	Tell MONIO device needs service.	2–42
SYSDMP	24	18	P	Emergency dump at end of job.	2–42
SYSCTL	25	19	P	Control Device.	2–40
SYSRSL	26	1A	S	Request next record from utility system tape and return.	2–42
SYSEOJ	27	1B	P	Processor or problem program final exit to monitor.	2–42
SYSDEB	28	1C	M	Call debug service routine.	2–42
SYSTRC	29	1D	M	Call trace service routine.	2–43
SYSRDS	30	1E	P	Call to IOPACK for general input.	2–34
SYSWRS	31	1F	P	Call to IOPACK for general output.	2–36
SYSWAT	32	20	P	Call for operator message and wait for response.	2–43
SYSIOI	33	21	U	Request monitor mode for IOPACK.	2–43
SYSIOO	34	22	U	Restore original mode after IOPACK.	2–43
SYSSTR	35	23	P	Set returns (EOF and error) for devices.	2–40
SYSPKY	36	24	P	Set protection key in PSW.	2–43
SYSSSK	37	25	P	Set storage protection key.	2–43
SYSPRS	38	26	P	Call to IOPACK to output print images.	2–37
SYSPUN	39	27	P	Call to IOPACK to output punch images.	2–39
SYSKMC	40	28	M	Set PSW for debugging system.	2–44
SYSDTF	41	29	S	Set up buffers for LISTIO.	2–44
SYSRAS	42	2A	M	Request specified record from utility system tape and exit to the record.	2–45

Table 2–5. Monitor Calls (Continued)

Mnemonic	Decimal Code	Hex.	Class*	Description	Page
SYSCLK	43	2B	P	Initialize timer and its interrupts.	2–32
SYSNAP	44	2C	P	Core snap for problem programs and system processors	2–46
SYSBRA	49	31	P	SVC assisted branch.	2–45
SYSIPR	240	F0	P	Initialize IOCE processor interrupt handling.	2–32
SYSRET	243	F3	P	Return from interrupt routine to caller.	2–42
SYSRTA	244	F4	P	Return from SVC interrupt routine to selected address (not to caller).	2–43
SYSDTH	245	F5	U	Terminate processing.	2–45
SYSCDP	246	F6	U	Request emergency dump.	2–45
SYSLGD	247	F7	U	Fatal logout on I/O device error.	2–45
SYSMOP	248	F8	M	Terminate I/O at end of job phase.	2–45
SYSLGC	249	F9	U	Fatal logout on I/O channel error.	2–45
SYSTIN	250	FA	U	Service console typewriter interrupt.	2–45
SYSTCE	251	FB	U	Record execution of I/O instruction.	2–45
SYSWRM	252	FC	U	Maintain count of occurrence of selected abnormal conditions.	2–46
SYSPIR	253	FD	U	Set initiate and interrupt routine addresses.	2–46
SYSTWR	254	FE	P	Write on console typewriter.	2–38
SYSTRE	255	FF	P	Read from console typewriter.	2–36

*Class symbols are as follows:

P—Unrestricted, including problem programs
U—Available to monitor and problem programs requesting USERIO
N—Available to system processors
S—Available to monitor and system processors
M—Monitor only

Misuse of SVC's classified U, N, S, or M normally results in a system dump.

2.5.4.4 External-Interrupt Routine

The external-interrupt routine handles interrupts caused by the interval timer or the console interrupt button within its own subsystem.

Other external interrupts result in logout and termination of processing. Depressing the Console Interrupt button on the operator control section of the system control panel results in an external interrupt. When this interrupt is recognized, the monitor types the message ENTER REQUEST to the operator. The operator may then respond with a CONT, END, TERM, PATC, PRIN, IPL, REIP, or SKIP message.

2.5.4.4.1 Timer Control. If the time value changes from positive to negative, the change causes an external interrupt.

The external-interrupt routine resets the timer to a specified value, sums the elapsed time for each job, and returns control to the next instruction. The address of this instruction is available in the old PSW instruction address field.

The capability exists to allow the programmer multiple timer interrupts, evenly spaced, from one initialization call.

The values are set and each time a timer interrupt is processed, the timer is reset to the user value specified. The calling sequence is:

```
SVC          SYSCLK
DC           XL4 'constant'
DC           AL4 (interrupt routine)
```

where

a. Constant is the timer value desired; it occupies a full word in the calling sequence but the actual value to set the clock is the first 3 bytes with the 4th byte ignored. The timer is expressed in 300ths of a second so the value should be the number of seconds before the interrupt multiplied by 300. When the programmer receives control after an interrupt, slightly more time than the time specified may have elapsed.

b. Interrupt routine is the address of the entry point for the programmer supplied routine.

The programmer may also provide his own interrupt routine for a single timer interrupt by using the following calling sequence:

```
SVC          SYSTIM
DC           XL4 'constant'
DC           AL4(interrupt routine)
```

where the parameters are the same as for SYSCLK.

The programmer-supplied routine is constructed according to the same rules as previously given for program interrupts.

2.5.4.4.2 IOCE Processor Interrupts. The programmer must provide his own interrupt routine for interrupts received by his execution in the IOCE Processor by using the following calling sequence:

```
SVC          SYSIPR
DC           AL4 (interrupt routine)
```

where interrupt routine is the address of the entry point for the programmer supplied routine.

The programmer-supplied routine is constructed according to the same rules as previously given for program interrupts.

2.5.4.5 Machine-Interrupt Routine

When the machine-interrupt mask bit is 1, a machine interrupt terminates the current instruction and transfers control to the emergency logout processor.

2.6 ANALYZING STORAGE DUMPS

This discussion is provided to aid programmers in debugging. The following information provides absolute storage addresses of interest to the user. The addresses are given in hexadecimal.

ADDRESS	CONTENTS
4F6	Name of last system record read
524	Last $ control card read
50	Timer
4FC	Date from $DATE card
5F0	Machine size in bytes.
664	First byte of available storage above program or processor (Double word aligned)

The following information is provided to aid users in debugging some of the more difficult program and hardware problems related to I/O. It is presented as a debug-oriented description of the most significant tables.

Note: In case of PSW restart dump, history is frozen at the time the restart button is depressed. In case of PRIN operator-requested dumps, I/O instruction history is frozen at the time of device-end following operator's typed address and length specification while I/O interrupts have been suppressed since the Attention key was depressed. In case of SYSDUMP, the history includes I/O operations which wrote initial dump records on .AUXIL.

2.6.1 Unit Assignment Table

A pointer at address 005FC gives the starting address of this 2-word-per-entry table, which is arranged in order by logical unit number. The contents of the bytes are as follows:

a. 0–1 Logical unit number (If 0, the unit does not exist for this job.)

b. 2–3 Physical device address

c. 4–7 Address of associated LISTIO.

2.6.2 LISTIO

LISTIO's are described in the Utility Monitor PLM. The most significant fields are the following:

a. Byte 48, bit 1 on and bit 3 off (xlx0xxxx) – Device is free, and no operation has been requested.

b. Byte 48, bit 2 is on (LIFG2) – Device is not ready, and the monitor expects a device-end interrupt when it becomes ready.

c. Bytes 64–65 (LIFLC) – Current file position for tapes.

d. Bytes 66–67 (LIBLC) – Current physical record position within file. (Positive numbers are measured from a start of file, negative numbers from end of file.)

e. Bytes 62–63 (LIDVA) – Physical device address.

2.6.3 IOPACK Register Save Table

This 16-word table at address 000700 contains registers 0–15, saved the last time IOPACK was called. Note the following:

a. Register 14 and 15 values are stored as if the call was BALR 14, 15, even for IOPACK routines called via SVC.

b. Register 1 save word may contain output value being returned to the caller.

c. SYSTWR and SYSTRE do not use this save area.

d. In SYSDUMP output, this area is set by SYSDUMP calls to IOPACK.

2.6.4 I/O Trace Table

This is a cyclic table of fifty 2-word entries, giving a coded history of I/O instructions executed. The table starts at address 000800, while 0007F8 contains a pointer to the last used entry. The 16 hexadecimal digits in an entry may be described as

IDDDSSSS CONNNNNN

where:

I is the instruction code

1 – SIO
2 – TIO
3 – HIO

DDD is the physical device address.

SSS is the device status (if presented by the device).

C is the condition code (for control operations, 1 with channel-end status on SIO indicates success; for data-transfer operations, SIO should invoke CC=0 response.)

O is the last four bits of the CCW operation code. The most common codes are:

1 or 9	**write**
2 or A	**read**
4	**sense**

NNNNNN is the count – how many times this exact instruction caused this exact response without the intervention of a different instruction or a different response.

2.6.5 I/O Interrupt Table

This is a cyclic table of fifty 4-word entries recording the I/O old PSW followed by CSW for the last 50 I/O interrupts. This table starts at address 000990; 007FC contains a pointer to the last used entry.

2.7 IOPACK CALLING SEQUENCES

The IOPACK section of the utility monitor consists of 18 routines which perform I/O functions for processors and object programs. These routines set end-of-file and error returns for I/O units, perform tape positioning and carriage control functions, read and write tape, read and punch cards, print, and perform system I/O functions. Ten of the routines each have a symbol that serves as an operand for register 15, which is normally used to transfer to these routines via the Branch and Link Register instruction (BALR). The other eight routines each have a symbol that serves as an operand for a supervisor call (SVC). These symbols are permanently defined in the assembler. A subprogram can use the symbols without defining them. Table 2–2 gives the IOPACK routines by class, their mode of call, their corresponding symbols, and their page of reference.

2.7.1 General-Input (SYSRDS) Routine

The general-input routine reads from tape or the on-line card reader according to options on $UNIT control cards. The SYSRDS routine also reads from the system input unit when logical unit 2 is called for and from the console typewriter when logical unit 12 is called for. Normally, it presents one logical record to the user in accordance with the data and count parameters and sets status according to actual record length transferred. In case of end-of-file or error, SYSRDS consults control information stored by the set-return routine and terminates the job or transfers to a user recovery routine, as appropriate.

The operation of SYSRDS described in the preceding sentence applies to all units except 2 and 12. Console typewriter problems and SYSIN errors are resolved within the monitor, and control is returned to the user only after successful I/O has occurred (i.e., Set-Return may not be used to change options for these two units). SYSIN dollar cards and EOFs are indicated by a $ in column 1 of the image. $END, $ENDTAPE, and 7/8EOF (delimiting $cards) return a record length of 1; all other $cards return a normal record length (i.e., currently 80). The user may read and process any $card(s) and continue to read the input stream :hrough any non-delimiting $card if he wishes. Trying to read past a delimiting $card or 7/8EOF will cause job termination, a diagnostic, and a SYSDUMP.

The calling sequence for the general-input routine is:

```
SVC     SYSRDS
DC      AL4(list)
```

where list is the address of the following out-of-line parameter list:

```
DC      F'xx'
DC      A(data)
DC      F'count'
```

where:

xx is the logical unit number of the input unit.

data is either the address of the data area or a register number. Any number less than 14 is assumed to be a register number, provided buffered operation is being used. With register specification, IOPACK leaves the record in the buffer and presents its address in the indicated register. (Register specification is not permitted for logical unit 2 or logical unit 12 or an on-line card reader.)

count specifies the number of data bytes to be transmitted. IOPACK never presents more than one logical record.

Register 1 normally is set to number of bytes transferred. However, on end-of-file or error it is used to transfer to the recovery routine.

2.7.2 System-Input-Unit (SYSJN) Routine

Note: The following routine is obsolete. Support for SYSJN has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to SVC SYSRDS on logical unit 2. The following description is provided only to aid in this conversion.

The system-input-unit routine reads from the system input unit. The system input unit is either a tape unit or the card reader depending on the assignment made by the operator.

If an object program or any system processor attempts to read any monitor control card or 7/8EOF card, a corresponding flag is set, the card is saved for later processing by the monitor, and in either case a $ is inserted in the first character of the card image presented to the user. If the user attempts to read the following card, control is transferred to the job control section of the monitor, which terminates the job.

The job is terminated if an uncorrectable read error is encountered on the system input unit.

The SYSJN calling sequence is different for problem programs and system processors.

The calling sequence for a System processor is:

```
L       15,SYSJN
CNOP    2,4
BALR    14,15
DC      A(data)
DC      A(eof)
```

where data is the address of the data area, and eof is the address of the processor's end-of-file routine.

The calling sequence for a problem program is:

```
L       15,SYSJN
CNOP    2,4
BALR    14,15
DC      A(data)
```

where data is the address of the data area.

2.7.3 Tape-Input (SYSRT) Routine

Note: The following routine is obsolete. Support for SYSRT has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to SVC SYSRDS on the desired logical unit. The following paragraphs are provided only to aid in this conversion.

The read-tape routine reads an input tape with or without buffering, depending on an option selected on the $UNIT control card. If buffers are requested, two buffers are used and initially filled when the first read-tape is executed. The size of the buffers is also specified on the $UNIT control card. When buffers are used, a call to the read-tape routine results in one of two operations, depending on the information specified in the second entry of the parameter list. If the address of a data area is given, the next logical record is moved to the data area; if a register number is given, the address of the next logical record is placed in the specified register and no data are moved. (Registers 14 and 15 cannot be used for this purpose, because they are used in linkage to IOPACK routines.) When one of the buffers is empty, it is filled with records from the input tape.

If buffers are not requested, the next record is read directly into the specified data area. If a register number is specified when no buffers are used, an error message is printed and the job is terminated. Control is returned to the calling program after the operation is completed.

Regardless of buffering specifications, the read-tape routine performs certain operations whenever it is used. The routine inserts the total number of bytes of the logical record into the parameter list. The high-order bit of the byte count is set to one if a byte-converter check occurred during the read operation. If blocking was specified on the $UNIT control card and the byte-converter check occurred when a blocked record was read, the high-order bit is set to one when the last logical record of the block is moved. The error return is taken if an uncorrectable error is encountered and all the logical records read up to that point have been moved to the data area. The tape is positioned at the end of the unreadable record when the error return is taken. The end-of-file return is taken when all the logical records have been moved to the data area and there are no more logical records to be read.

The calling sequence for the read-tape routine is:

```
L       15,SYSRT
CNOP    2,4
BALR    14,15
DC      A(list)
```

where list is the address of the following parameter list:

```
DC      F'xx'
DC      A(data)
DC      F'count'
```

where:

xx is the 2-digit logical unit number of the input tape.

data is either the address of the data area or a register number. Any number less than 14 is considered a register number. When a register is specified, IOPACK places the address of the record in the specified register.

count contains the total number of bytes in the logical record. This count is supplied by IOPACK.

Note: The parameter list must be out of line because the routine returns control to the instruction immediately after the DC in the calling sequence.

2.7.4 On-Line Card Reader (SYSRC) Routine

Note: The following routine is obsolete. Support for SYSRC has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to SVC SYSRDS where the 2-digit logical unit number of the card reader is provided on the $UNIT control card. The following paragraphs are provided only to aid in this conversion.

This routine allows the programmer to use an on-line card reader that is not serving as the system input unit. The read-card routine moves a card image from the buffer to the specified data area and initiates a Card Read command to read the next card into the buffer. Card reading is a single-buffered operation. The error return is taken if an uncorrectable error is encountered and the last card in the buffer has been moved to the data area; the end-of-file return is taken after the last card has been moved to the data area.

The calling sequence for the read-card routine is:

```
L        15,SYSRC
CNOP     2,4
BALR     14,15
DC       A(list)
```

where list is the address of the following parameter list:

```
DC       F'xx'
DC       A(data)
```

where:

xx is the 2-digit logical unit number of the card reader.

data is the address of the data area (80 bytes).

Note: The parameter list must be out of line, because the routine returns control to the instruction immediately after the DC in the calling sequence.

2.7.5 Console-Typewriter-Input (SYSTRE) Routine

The read-console-typewriter routine reads the information typed into the console typewriter. This operation is unbuffered. When the information has been read, control is returned to the instruction immediately after the second DC statement in the calling sequence.

The calling sequence for the read-console-typewriter routine is:

```
SVC      SYSTRE
DC       AL3(data)
DC       AL1(count)
```

where:

data is the address of the area in which the information is to be placed.

count is the number of characters to read (minimum 2, maximum 132).

Note: The 1052 read buffer is not re-initialized via the CANCEL key. The following example illustrates the point.

Operator types	123456
Operator hits CNACEL key	
Operator types	9876
Operator depresses ENTER key	
Program finds in 1052 read area	987656

To prevent this undesired situation, the operator must enter trailing blanks if the proper response is shorter than the incorrect response which was CANCELed.

2.7.6 General-Output (SYSWRS) Routine

The general-output routine outputs data records to tape, to the on-line printer with single-spacing, and to the on-line punch according to options selected on $UNIT cards. The SYSWRS routine also outputs to the console typewriter when logical unit 12 is called for. The general-output routine accepts one logical record from the user on each call in accordance with the data and count parameters and sets status (contained in register 1) to zero except as follows. In case of end-of-tape or error, SYSWRS consults information stored by the set-return routine and information from $REEL cards and either issues a message to the operator to mount a new reel, terminate the job, or transfers to a user recovery routine.

The calling sequence for the general-output routine is:

```
SVC      SYSWRS
DC       AL4(list)
```

where list is the address of the following out-of-line parameter list:

```
DC       F'xx'
DC       A(data)
DC       F'count'
```

where:

xx is the logical unit number of the output unit.

data is the address of the data area.

count is the number of data bytes to be transmitted.

In case of failure of the operation or end-of-tape register 1 is used to transfer to a recovery routine.

Note: On PRT format tapes (SYSOUT), SYSWRS prefixes the record with a blank as it assumes the caller has not included carriage control but wants to print single space.

2.7.7 Tape-Output (SYSWT) Routine

Note: The following routine is absolete. Support for SYSWT has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to the general-output routine (SVC SYSWRS). The following paragraphs are provided only to aid in this conversion.

The write-tape routine writes an output tape with or without buffering, depending on the option selected on the $UNIT control card. If buffers are requested, two buffers are used, and a call to the write-tape routine causes a logical record to be moved from the data area to an output buffer. When one of the buffers is full, the contents of the buffer are written on the output tape. If buffers are not requested, the logical record is written onto the tape directly from the data area. Control is returned to the calling program when the operation is complete and checked.

The error return is taken when an uncorrectable error is encountered. If no number is present or if the list of numbers on the $REEL control card is exhausted, the end-of-file return is taken.

If the $REEL control card requested file protection for this file, no attempt is made to write on the tape. An error message is issued, and the job is terminated.

The calling sequence for the write-tape routine is:

```
L       15,SYSWT
CNOP    2,4
BALR    14,15
DC      A(list)
```

where list is the address of the following parameter list:

```
DC      F'xx'
DC      A(data)
DC      F'count'
```

where:

xx is the 2-digit logical unit number of the tape.

data is the address of the data area.

count is the number of bytes in the logical record if variable-length blocked or unblocked; if the records are fixed-length blocked or unblocked, the count is specified on the $UNIT control card and need not be given here.

Note: The parameter list must be out of line, because the routine transfers control to the instruction immediately after the DC in the calling sequence.

2.7.8 General-Print (SYSPRS) Routine

The general-print routine outputs print images to tape, to an on-line printer, and to an on-line punch according to options selected on $UNIT cards. The SYSPRS routine also outputs to the system output unit when logical unit 3 is called for and to the console typewriter when logical unit 12 is called for. When writing on tape, SYSPRS assumes in each call the user has presented one logical record, but in outputting to on-line devices, SYSPRS assumes the user may have blocked several records together using record marks (X'E0') to separate the individual logical records. The SYSPRS routine processes end of tape, uncorrectable error conditions, and successful operation in the same manner as the SYSWRS routine.

When writing on 9-track tape, SYSPRS writes the data exactly as presented, blocking records in standard 9020 format if specified.

When writing on 7-track tape, SYSPRS proceeds according to $UNIT card options. Note that this will not normally produce usable results since PRT record format cannot be declared on $UNIT cards but is only declared internally by the monitor when appropriate for SYSOUT.

In outputting to the on-line printer, SYSPRS unblocks records according to record marks and prints each individual record as a separate image, using the first character for carriage control. Records may be 2 to 133 characters long including the carriage control byte and are positioned according to carriage control character as follows (there is no space after printing):

Character	Spacing
+,&	No space before printing (overprint)
blank	1 space before printing
0	2 spaces before printing
–	3 spaces before printing
1–5	Skip to channel 1–5 before printing

In outputting to an on-line punch, SYSPRS unblocks records according to record marks, inspects the first character of each individual record (assumed to be the carriage control byte) and punches the remainder (up to 80 characters) in pocket 3 if the carriage control character is a W, pocket 2 otherwise.

In outputting to the console typewriter, SYSPRS unblocks records according to record marks and prints each record (up to a limit of 132 characters per record).

The calling sequence for the general-print routine is:

```
SVC     SYSPRS
DC      AL4(list)
```

where list is the address of the following out-of-line parameter list:

```
DC      F'xx'
DC      A(data)
DC      F'count'
```

where:

xx is the logical unit number of the output unit.

data is the address of the data area.

count is the number of data bytes to be transmitted.

In case of failure of the operation or end of tape register 1 is used to transfer to a recovery routine.

2.7.9 System-Output-Unit-Print (SYSJT) Routine

Note: The following routine is obsolete. Support for SYSJT has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to the general-print routine (SVC SYSPRS) on logical unit 3 (the system output unit). The following paragraphs are provided only as an aid in this conversion.

The job-print routine writes 132-character records from a data area on either a tape unit or the on-line printer according to the unit assignment of the system output unit (.PRINT). If the unit specified is a tape unit, the records are blocked. The job is terminated if an uncorrectable error is encountered. The sequence is as follows:

```
          L         15,SYSJT
          CNOP      0,4
          BALR      14,15
          DC        H'cc'
          DC        A(data)
```

where:

cc specifies one of the following carriage-control code numbers:

0 – Suppresses spacing
1 – Spaces one line after printing
2 – Spaces two lines after printing
3 – Spaces three lines after printing
4 – Skips to channel 2 after printing
5 – Skips to channel 3 after printing
6 – Skips to channel 4 after printing
7 – Skips to channel 5 after printing
9 – Skips to new page after printing
11 – Spaces one line immediately
12 – Spaces two lines immediately
13 – Spaces three lines immediately
14 – Skips to channel 2 immediately
15 – Skips to channel 3 immediately
16 – Skips to channel 4 immediately
17 – Skips to channel 5 immediately
19 – Skips to new page

data is the address of the data area. One print line is written each time SYSJT is called. The data parameter is required, although its contents are ignored when the carriage control number is 11 to 19.

2.7.10 On-Line-Printer (SYSPRT) Routine

Note: The following routine is obsolete. Support for SYSPRT has been scheduled to be immediately removed from the assembler. All existing program references should be changed to the general print routine (SVC SYSPRS) with the 2-digit logical unit number of the printer as specified on the $UNIT card. The following description is retained only as an aid to this conversion. MONITOR support for SYSPRT will remain for BATCH.

This routine allows the programmer to use an on-line printer that is not serving as the system output unit. The print routine moves 132 characters from the data area to the output buffer and initiates the print operation. The error return is taken if an uncorrectable error is encountered.

The calling sequence for the print routine is:

```
          L         15,SYSPRT
          CNOP      2,4
          BALR      14,15
          DC        A(list)
```

where list is the address of the following parameter list:

```
          DC        F'xx'
          DC        A(data)
          DC        F'cc'
```

where:

xx is the 2-digit logical unit number of the printer, as specified on the $UNIT control card.

data is the address of the data area, unless the carriage control code is 11, 12, or 13, in which case it is all zeros.

cc specifies a carriage-control number from 0 to 19, as listed in paragraph 2.7.9.

Note: The parameter list must be out of line, because the routine returns control to the instruction immediately after the DC in the calling sequence.

2.7.11 Console-Typewriter-Output (SYSTWR) Routine

The write-console-typewriter routine types information on the console typewriter from a data area. This operatior is a single-buffered operation. When the information has been typed, control is returned to the instruction immediately after the second DC statement in the calling sequence.

The calling sequence for the write-console-typewriter routine is:

```
SVC     SYSTWR
DC      AL3(data)
DC      AL1(count)
```

where:

data is the address of the data area containing the information to be typed.

count is the number of characters to be written (minimum 2, maximum 132).

2.7.12 General-Punch (SYSPUN) Routine

The general-punch routine outputs punch images to tape and to the on-line punch according to options selected on the $UNIT cards. The SYSPUN routine also outputs to the system output unit when logical unit 4 is called for and to the console typewriter when logical unit 12 is called for. The SYSPUN routine processes end-of-tape, uncorrectable error conditions and returns status in the same manner as SYSWRS. The general punch routine never assumes the user has done any blocking within his program and processes data as follows to permit SYSPRS and SYSPUN to output to common files.

When writing on 9-track tape, SYSPUN prefixes each record with a character "V" to identify it as a punch record, pads it to a total length of 81 characters, and blocks records in standard 9020 format if specified.

~~When writing on 7-track tape, SYSPUN proceeds according to $UNIT card options. Note that this will not normally produce usable results since PRT record format is appropriate (and selected by the monitor for SYSOUT) but cannot be declared on $UNIT cards.~~

When writing on the on-line punch, SYSPUN punches the record exactly as presented (up to 80-characters). When writing on the console typewriter, SYSPUN prints the record exactly as presented (up to 80 characters).

The calling sequence for the general-punch routine is:

```
SVC     SYSPUN
DC      AL4(list)
```

where:

```
DC      F'xx'
DC      A(data)
DC      F'count'
```

where:

xx is the logical unit number of the output unit.

data is the address of the data area.

count is the number of data bytes to be transmitted.

In case of failure of the operation or end of tape, register 1 is used to transfer to a recovery routine.

2.7.13 System-Output-Unit-Punch (SYSJH) Routine

Note: The following routine is obsolete. Support for SYSJH has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to the general punch routine (SVC SYSPUN) where the logical unit number of the output unit is 4 (system punch output). The following description is retained only as an aid in this conversion.

The job-punch routine either punches cards or writes blocked card images on tape from a data area, depending on the unit assignment of the system output unit (.PUNCH). The job is terminated if an uncorrectable error is encountered.

The calling sequence for the job-punch routine is:

```
L       15,SYSJH
CNOP    2,4
BALR    14,15
DC      A(data)
```

where data is the address of the data area. One card image is written each time SYSJH is called.

2.7.14 On-Line-Punch (SYSPC) Routine

Note: The following routine is obsolete. Support for SYSPC has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to the general punch routine (SVC SYSPUN) with the 2-digit logical unit number of the on-line punch as specified on the $UNIT card. The following description is retained as an aid in this conversion.

This routine allows the programmer to use an on-line punch that is not serving as the system output unit. The punch-card routine moves a card image from the data area to the output buffer and initiates the punch operation. Card punching is a single-buffered operation.

The error return is activated if an uncorrectable error is encountered.

The calling sequence for the punch-card routine is:

```
        L       15,SYSPC
        CNOP    2,4
        BALR    14,15
        DC      A(list)
```

where list is the address of the following parameter list:

```
        DC      F'xx'
        DC      A(data)
```

where:

xx is the 2-digit logical unit number of the card punch.

data is the address of the data area.

Note: The parameter list must be out of line, because the routine returns control to the instruction immediately after the DC in the calling sequence.

2.7.15 Set-Return (SYSSTR) Routine

The set-return routine furnishes IOPACK with end of file (end of tape) and error return addresses for each unit being used in a job. These addresses are expected for each unit declared on a $UNIT card. For limited-use programs, however, the programmer may prefer to have IOPACK terminate the job in case of an unexpected end of file or uncorrectable error. The error return is the address of a routine to which control should be transferred if an uncorrectable error is discovered; the end-of-file return is the address of a routine to which control should be transferred when an end-of-file (when reading) or an end-of-reel (when writing) is encountered.

The calling sequence for the set-return routine is:

```
        SVC     SYSSTR
        DC      AL4(list)
```

and LIST is the out-of-line parameter list:

```
LIST    DC      F'logical unit'
        DC      A(end-of-file routine)
        DC      A(error routine)
```

end-of-file is the address of a routine to handle either of the following two situations:

1. End-of-file (when a tapemark is encountered during a read operation).
2. End-of-reel encountered while writing when no more reels are specified on $REEL control card.

Note 1: If the return address contains a zero (i.e., DC F'0') and the exit is taken, a diagnostic is printed and a SYSDUMP is taken. THIS IS DEFAULT.

Note 2: If the return address contains a one (i.e., DC F'1') and the exit is taken, a diagnostic will be printed and the job terminated WITHOUT a dump.

Note 3: In transferring to a recovery routine, the monitor restores registers 2–15 as for a normal return and loads register 1 with the address of the recovery routine for use as a base register.

2.7.16 Unit-Control (SYSCTL) Routine

The unit-control routine performs one of a variety of control operations. The operations that the user may request with a single call are as follows:

a. Backspace block. (This operation is only valid following read or backspace block. Following read, it will backspace the tape one physical record for unbuffered operation and two physical records for buffered operation and then "close" the file so that the next read operation reads the next logical record. Following backspace block, the operation backspaces the tape one physical record.)

b. Backspacing files. (At the completion of this operation, the tape is positioned ready to read the tapemark that stopped the operation.)

c. Backspacing records. (This operation is valid only if buffering has not been requested.)

d. Forward-spacing files. (At the end of this operation, the tape is positioned after the tapemark that stopped this operation. If end-of-tape is encountered, it is treated as an uncorrectable error.)

e. Forward-space record. (This operation is valid only if buffering has not been requested.)

f. Rewinding files.

g. Rewinding and unloading files.

h. Writing tapemarks. (If buffers are being used, the contents of the buffers are placed on the tape before the tapemark is written. If the $REEL control card requested file protection, no attempt is made to write the tapemark; an error message is issued, and the job is terminated.)

i. Writing tapemarks on, and rewinding files. (The qualifications for item h also apply to this operation.)

j. Writing tapemarks on, and rewinding and unloading files. (The qualifications for item h also apply to this operation.)

The end-of-file return is taken if end of file is sensed during a forward-space or backspace record or block operation. The error return indicated via SYSSTR is taken in case of uncorrectable error.

The calling sequence for the tape-control routine is:

```
        SVC     SYSCTL
        DC      AL4(list)
```

where list is the address of the following parameter list:

```
DC      F'xx'
DC      CL4'mnemonic'
```

where:

xx is the 2-digit logical unit number of the tape involved in the operation.

mnemonic is one of the following, depending on the operation desired:

Mnemonic	Operation
BSB	Backspace block
BSF	Backspace file
BSP	Backspace record
FSF	Forward-space file
FSP	Forward-space record
RWD	Rewind
RUN	Rewind and unload
WTM	Write tapemark
TMR	Write tapemark and rewind
TRU	Write tapemark, rewind, and unload

Note: The parameter list must be out of line because the routine returns control to the instruction immediately after the DC in the calling sequence.

2.7.17 Set-Return (SYSSR) Routine

Note: Support for SYSSR has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to SVC SYSSTR. One SVC SYSSTR call must be provided for each unit for which returns are being supplied.

This is an obsolete routine which functions identically to SYSSTR. The only difference is the calling sequence which is:

```
L       15,SYSSR
CNOP    0,4
BALR    14,15
DC      H'yy'
DC      F'xx'            ⎫
DC      A(end of file)   ⎬ repeated yy times
DC      A(error)         ⎭
```

where:

yy is the total number of units for which returns are being supplied.

xx is the logical unit number.

other values are as for SYSSTR.

2.7.18 Unit-Control (SYSCT) Routine

Note: Support for SYSCT has been scheduled to be immediately removed from both the assembler and the monitor. All existing program references should be changed to SVC SYSCTL.

This is an obsolete routine which functions identically to SYSCTL. The only difference is the calling sequence which is:

```
L       15,SYSCT
CNOP    2,4
BALR    14,15
DC      A(list)
```

where list is the same as for SYSCTL.

2.8 MONITOR CALLS

This section contains an explanation of all monitor supervisor calls which are available to the problem-programmer.

2.8.1 SYSPIN (Initialize Program Interrupts)

This SVC is discussed in paragraph 2.5.4.1.

2.8.2 SYSCOM (System Processor – Monitor Communications)

Use of this SVC is restricted to system processors not permitted to store directly into monitor communications tables (including jobs being run as MOD = SYST).

The required calling sequence is:

```
        CNOP    2,4
        SVC     SYSCOM
        DC      A(args)

ARGS    DC      F'flags'
        DC      CL2'  '
        DC      CL6'name'

            or alternate form

        DC      F'flags'
        DS      F'unused'
        DC      F'new SYSDUMP start address'
        DC      F'new SYSDUMP end address'
```

where:

ARGS is the address of an out-of-line argument area.

FLAGS indicates action as described later.

NAME is the name of the required system tape record (when appropriate), or the new SYSDUMP start address preceded by a full word of zeros.

VAL – new end address for SYSDUMP.

SYSCOM will take one of the following actions according to the value of FLAGS:

x'0001' Read record "NAME" from the system tape, and exit to it.

x'0002' Read record "NAME" from the current file on the system tape, and return to caller.

x'0006' Same as 0002, but read patch cards from SYSIN before entering the new processor.

x'0005' Same as 0001, but read patch cards from SYSIN before entering the new processor.

x'0008' Modify SYSDUMP limits (i.e., start and end dump limits).

x'0100' Processor has successfully completed operation.

x'0200' Processor was unable to complete successfully, and anything written on .AUXIL is meaningless. The job may be continued as a nonexecute job.

x'0400' Because of serious problems, the job must be terminated; however, a storage dump is not justified.

x'0800' Because of serious problems (probable program error), the job should be terminated with a system dump.

x'1000' Because of catastrophic problems, the batch should be terminated; however, a storage dump is not justified.

x'2000' Because of catastrophic problems (probable machine failure or monitor error), the batch should be terminated with an on-line emergency dump.

Note: Contents of GPRs (General Purpose Registers) not transparent across SVC SYSCOM.

2.8.3 SYSTIM/SYSCLK (Set Up Interval Timer Routine)

This SVC is discussed in paragraph 2.5.4.4.

2.8.4 SYSIO (Tell MONIO Device Needs Service)

This SVC is available to IOPACK and comparable USERIO routines to notify MONIO of the need for service. The address of the LISTIO controlling the device must be stored in CRCHPT before executing the SVC. MONIO will start the device (unless the channel is already busy) and then return.

2.8.5 SYSDMP (Request Emergency Dump)

This SVC will cause the monitor to terminate the job and dump storage in accordance with emergency dump debug cards, or dump storage if the job had not DUMPE debug cards.

2.8.6 SYSRET (Return to Caller)

Though available to all users, this SVC must be used only to return control from SVC routines for which the specified exit is SYSRET or SYSRTA. SYSRET returns to the instruction following original SVC.

2.8.7 SYSRSL (Request Record from NOSS system tape and Return)

This SVC is available only to system processors transferred from the utility system since its design violates storage-protect rules.

2.8.8 SYSEOJ (Return at End of Processor or Problem Program)

This SVC is available to all users for final return to the monitor.

2.8.9 SYSDEB (Call Debug Subroutine)

This SVC is restricted to debugging system use, although the loader will insert this SVC in problem programs being loaded with debug requests.

2.8.10 SYSTRC (Call Trace Subroutines)

This SVC is subject to the same rules as SYSDEB.

2.8.11 SYSRDS (Call IOPACK for General Input)

This SVC is discussed in paragraph 2.7.1.

2.8.12 SYSWRS (Call IOPACK for General Output)

This SVC is discussed in paragraph 2.7.6.

2.8.13 SYSWAT (Call for Operator Message and Wait for Response)

This SVC has only limited utility since it will always type "PROGRAM WAITING" and always require an operator response of "CONT."

2.8.14 SYSIOI (Request Monitor Mode for IOPACK)

This SVC is available to IOPACK and comparable USERIO routines to ask that the PSW be changed to Monitor mode and assigned a zero-protect key.

2.8.15 SYSIOO (Restore Original PSW Mode)

This SVC is used by IOPACK to restore original mode (monitor/problem) and original storage-protect key. Other programs may use this SVC after SYSIOI only if they have not made intervening calls to IOPACK.

2.8.16 SYSRTA (Return to Selected Address)

Though available to all users, this SVC must be used only to return control from SVC routines for which the specified return is SYSRET or SYSRTA.

SYSRTA assumes a calling sequence of the form:

```
SVC     SYSRTA
DC      AL4(ADR)
```

where ADR is the address to which to transfer control.

2.8.17 SYSPKY (Set Protect Key in PSW)

This SVC changes the storage-protect key in the PSW. At the start of execution, this key is set to 0010. However, if the programmer has changed the storage-protect key for a block of storage, the key in the PSW must also be changed to the same value set for the block if it is intended to store data in that block. The following calling sequence is used to change the storage-protect key in the PSW:

```
SVC     SYSPKY
DC      H'xx'
```

where xx is the new storage-protect key (in decimal, from 2–15). The monitor replaces bits 8–11 of the PSW with the storage-protect key requested in the calling sequence. However, it first checks to see that the requested key is not 0000 or 0001, which would permit the user control over the complete system.

2.8.18 SYSSSK (Set Storage Protect Key)

The following calling sequence is used to change the storage-protect key in any storage block(s) outside the monitor area:

```
CNOP    2,4
SVC     SYSSSK
DC      H'xx'
DC      H'yy'
DC      A(zz)
```

where:

xx is the new storage-protect key (0–15 if store-protect only is desired, 16 + desired key value if store and fetch protection are both desired).

yy is the number of consecutive blocks of 2048 bytes each.

zz is the starting address.

At execution time, the monitor sets the storage-protect key for the number of blocks specified. It checks to see that the starting address given has four low-order zeros (to avoid a specification interrupt) and that it is not within the monitor area. It also checks to see that each block to be given a storage-protect key is within the limits of the particular machine (to avoid an addressing interrupt). If any of these tests fail, a diagnostic message is produced and the job is terminated.

2.8.19 SYSPRS (Call IOPACK to Output Print Images)

This SVC is discussed in paragraph 2.7.8.

2.8.20 SYSPUN (Call IOPACK to Output Punch Images)

This SVC is discussed in paragraph 2.7.12.

2.8.21 SYSKMC (Set PSW for Debugging System)

This SVC is restricted to debugging system use.

2.8.22 SYSDTF (Set Up Buffers for LISTIO)

This SVC is restricted to use by system processors. To use, buffers must be assigned within the processors' storage area, the device must be free (not busy and no waiting request for operation) and the appropriate calling sequence must be used. Return is in line.

To specify variable blocked records, the following calling sequence is used:

```
SVC     SYSDTF
DC      AL2(logical unit)
DC      AL4(address of buffer 0)
DC      AL4(address of buffer 1)
DC      AL4(address oof buffer 0 + 2)
DC      AL4(address of buffer 0 + 4)
DC      AL4 (0)
DC      X'04'
DC      AL2(buffer size)
DC      AL1(1)
DC      AL1(3)
DC      X'mode command for 7-track tape'
```

Note: Maximum buffer size for VBL records is 32767 (i.e., X'7FFF')

For variable-length records, the following calling sequence is used:

```
SVC     SYSDTF
DC      AL2(logical unit)
DC      AL4(address of buffer 0)
DC      AL4(address of buffer 1)
DC      AL4(address of buffer 0)
DC      AL4 (0)
DC      AL4 (0)
DC      X'04'
DC      AL2(buffer size)
DC      AL1 (1)
DC      AL1 (2)
DC      X'mode command for 7-track tape'
```

For fixed-length blocked records, the calling sequence is:

```
SVC     SYSDTF
DC      AL2 (logical unit)
DC      AL4 (address of buffer 0)
DC      AL4 (address of buffer 1)
DC      AL4 (address of buffer 0)
DC      AL4 (address of buffer 0 + logical
            record length)
DC      AL4 (logical record length)
DC      X'04'
DC      AL2 (buffer size)*
DC      AL1 (blocking factor)
DC      AL1 (1)
DC      X'mode command for 7-track tape'
```

*Logical record length multiplied by blocking factor.

For fixed-length records, the calling sequence is:

```
SVC     SYSDTF
DC      AL2 (Logical unit)
DC      AL4 (address of buffer 0)
DC      AL4 (address of buffer 1)
DC      AL4 (address of buffer 0)
DC      AL4 (0)
DC      AL4 (logical record length)
DC      X'04'
DC      AL2 (buffer size)
DC      AL1 (1)
DC      AL1 (0)
DC      X'mode command for 7-track tape'
```

For unblocked unbuffered records, the calling sequence is:

```
SVC     SYSDTF
DC      AL2 (logical unit)
DC        AL4 (0)
DC        AL4 (0)
DC        AL4 (0)
DC        AL4 (0)
DC        AL4 (0)
DC        X'00'
DC        AL2 (0)
DC        AL1 (1)
DC        AL1 (2)
DC        X'mode command for 7-track tape'
```

2.8.23 SYSRAS (Request Specified Record from NOSS System Tape)

This SVC is available only to system processors transferred from the utility system since its design violates storage-protect rules.

2.8.24 SYSBRA (Monitor-Assisted Program Transfer)

This SVC is unrestricted but is intended for use in program patches.

SYSBRA assumes a calling sequence of the form:

```
SVC       SYSBRA
DC        AL4 (adr)
```

where ADR is the address to which to transfer control.

2.8.25 SYSDTH (Terminate Processing)

This SVC is restricted to monitor use.

2.8.26 SYSCDP (Request Emergency Dump)

This SVC is restricted to monitor use.

2.8.27 SYSLGD (Fatal Logout on I/O Device Error)

This SVC is restricted to monitor and USERIO routines which have detected an I/O device error which is serious enough to justify terminating processing with a logout message.

SYSLGD assumes a calling sequence of the form.

```
 SVC     SYSLGD

 DC      XL'0'        STATUS BYTE

 DC      XL3'0'       DEVICE ADDRESS 12 BIT
                      (RIGHT JUSTIFIED)

 DC      XL8'0'       CSW

DC       XL8'0'       CCW

 DC      X'0...0'     SENSE BYTES ONE OR MORE
```

2.8.28 SYSMOP (Terminate I/O at End-of-Job Phase)

This SVC is restricted to monitor use.

2.8.29 SYSLGC (Fatal Logout on I/O Channel Error)

This SVC is restricted to monitor and USERIO routines which have detected an I/O channel error which is serious enough to justify terminating processing with a logout message.

SYSLGC assumes a calling sequence of the form:

```
SVC     SYSLGC

DC      XL'0'        CHANNEL STATUS BYTE

DC      XL3'0'       DEVICE ADDRESS 12 BITS
                     (RIGHT JUSTIFIED)
```

2.8.30 SYSTIN (Service Console Typewriter Interrupt)

This SVC is restricted to monitor use.

2.8.31 SYSTCE (Record Execution of I/O Instruction)

This SVC (code 251) is available to USERIO and should immediately precede every machine I/O instruction. SYSTCE records information as to what was attempted and the resulting condition code and CSW, but returns control to the user as if the instruction had been executed in line. (Executed instructions may not use register 11, 14, 15 as base.)

2.8.32 SYSWRM (Count Occurrence of Selected Abnormal Conditions)

This SVC (code 252) is intended for monitor use but is not restricted. The routine maintains a table of locations where SVC SYSWRM has been executed with a count of number of times for each location. The table is not readily accessible to users because it is not at a fixed location within the monitor.

2.8.33 SYSPIR (Set Initiate and Interrupt Routine Addresses)

This SVC is available only to programs using USERIO and is used to inform the monitor of entry points to user-service routines comparable to MONIO, for use with device types 6 through 10. SYSPIR assumes a calling sequence of the form:

```
CNOP    2,4
SVC     SYSPIR
DC      A(inter)   type 06
DC      A(start)
DC      A(inter)   type 07
DC      A(start)
DC      A(inter)   type 08
DC      A(start)
DC      A(inter)   type 09
DC      A(start)
DC      A(inter)   type 10
DC      A(start)
```

where:

start is the address of the routine that starts IO for the device type (0 if not used.)

inter is the address of the routine that services interrupts for the device type (0 if not used).

2.8.34 SYSTWR (Write on Console Typewriter)

This SVC is discussed in paragraph 2.7.11.

2.8.35 SYSTRE (Read from Console Typewriter)

This SVC is discussed in paragraph 2.7.5.

2.8.36 SYSNAP (Core Snap for Programs and Processors)

This SVC can be used by any program (problem program or system processor) to dump the general purpose registers and one area of core storage to SYSOUT.

SYSNAP assumes a calling sequence of the form:

```
SYSNAP  EQU    44

        SVC    SYSNAP
        DC     AL4 (LIST)
```

where LIST is the out-of-line parameter list:

```
LIST    DC     A(Dump Start)
        DC     A(Dump End)
```

Although this service is extremely useful in development testing, its prime use is anticipated to be in the area of PTR analysis and verification where it will be invoked via REPs.

Constraints:

a. EQU required in source usage
b. Output is Hex only
c. Duplicate lines are not bypassed
d. Cannot be used to dump the last eight words of core storage.

2.9 USER MESSAGES

Messages generated by the Utility System are classified in three groups:

1. Normally of interest only to the computer operator. These are outputted via the console typewriter and described in the Utility System Operator's Manual.

2. Normally of interest only to the computer user. These are outputted via the system output unit (SYSOUT) and described below.

3. Normally of interest to both the operator and the user. These are outputted via both SYSOUT and the console typewriter, are described below and in the Utility System Operator's Reference Manual, and are flagged with an asterisk below.

The user messages are as follows:

* **xxxxxx NOT A SYSTEM PROCESSOR**

Either the preceding $control card specifies a processor name which is not on the system tape or the card is at an incorrect position in the job deck. Job processing will skip to the next $ control card, and execution will not be permitted.

* **– – OUT OF SEQ OR INV CD – –**

Either a non-$control card was read when a $control card was required or a $XEQ, $NONEX, or $AUX card has errors in its option fields.

In case of a non-$control card, the job is terminated. Errors in $XEQ, $AUX, or $NONEX option fields terminate scanning of the card but do not otherwise affect job operation.

Note: The entire card is printed with the overlaying columns 40–65 of the card in error.

BAD ENV CARD

The preceding $ENV card has errors in grammar. Any fields not already processed will be ignored.

* **BAL**

Either the assembler has been called via a $BAL card, or JOVIAL or SYMCOR processing is complete and the BAL assembler is being called to assemble their output.

* **DEBEDT**

Debugging output was placed on .AUXIL during problem program execution and/or SYSDUMP output has been placed there and/or tape dump debug cards were included in the object deck. The edit portion of the debugging system is being called to reduce and print this data.

ELAPSED TIME 00/00/00

Time in hours, minutes, and seconds required to process the last job. (Time is measured from IPL to end of first job and from end of previous job to end of current job.)

* **END OF BATCH – PT.HH/mm L.xxxxxx C.xxxxx**

Current batch of jobs has completed. If SYSOUT is online, only "END OF BATCH" is printed. If SYSOUT is to tape, the entire message is printed. Content of the new fields is as follows:

PT. HH/mm	–	Estimated print time for this batch assuming a high-speed printer (i.e. 1100 Lpm). If a low-speed printer is to be used for printing, double this estimated print time.
L.xxxxxx	–	Lines output to SYSOUT during this batch.
C.xxxxx	–	Punch cards output to SYSOUT during this batch.

* **END OF JOB**

Job is finished.

EOF OR $END ON SYSIN

A program has requested a record to be read from the system input unit. The previous record delivered to the program or system processor was a $END, $ENDTAPE or 7/8EOF control card and the program was expected to do final housekeeping and return control to the monitor without requesting more input from SYSIN. (For easy recognition, the image presented contained a $ as its first character and a record length of 1.) The request is ignored, the message is printed as part of the output, and the job is terminated with a SYSDUMP.

* **ERR COND 3**

A processor has returned control to the utility monitor, but all processor classification flags are off.

* **ERR ON BASIC UNIT**

The system has encountered an error or unexpected end of file on a basic system unit. The job will be terminated.

* **ID MISSING**

The $ID control card for this job is either missing or out of place. The system generates a blank name for the job and continues processing.

INVALID IO LINKAGE

An IOPACK linkage has been found to be in error because of improper boundary alignment of parameters, improper data address or byte count on output operations, or incorrect linkage register loading. A system dump is taken, and the job is terminated. Registers 14 and 15, appearing on the dump, may be consulted to determine the location of the bad linkage.

* **UNEXPECTED EOF, EOT, or ERROR**

An unexpected end-of-file was encountered while reading, or an unexpected end-of-tape was encountered while writing, or a permanent I/O error has been encountered (to which the operator typed in CONT). To eliminate this diagnostic (and dump) initialize EOF/ERR Returns with SVC SYSSTR.

* **LISTGN**

The LISTIO generator has been called to assign devices for $UNIT cards.

* **USERIO LOG UNT ERROR**

A system processor $UNIT card has an invalid logical unit number specified. Job will be terminated.

* **USERIO CRD TYP ERROR**

A processor $UNIT card does not correctly indicate USERIO. Job is terminated.

* **USERIO DEV TYP ERROR**

Illegal device type in a processor USERIO $UNIT card. The job will be terminated.

* **USERIO DEV ADR ERROR**

The device address specified on a processor $UNIT card is not legal, or is invalid. Termination of job.

* **DUP USERIO L.U. NUM OR DEV ADR, CARD IGNORED**

The logical unit number or the device address on a processor USERIO $UNIT card duplicates that of an existing system unit. Card is ignored.

* **SYSTEM USERIO L.U. XX – ACCEPTED**

A non-standard I/O device is available for use by a system processor, and has been defined as a system unit.

* **SYSTEM USERIO L.U. XX NOT AVAILABLE, CONTINUING**

A requested non-standard I/O device is not available to processor, but job will continue.

* **LOADING**

Loading has begun.

* **LOADING ERROR NO XEQ**

The loader has found an error while loading. Execution has been suppressed.

* **SUPPRESSED LOADING**

Because of previous indicated errors, or because of operator request, the job will not be executed.

LOG UNIT xx NOT IN ASSGNMNT TBL

An I/O operation has been requested on logical unit xxx, but this unit cannot be located in the assignment table. (This condition occurs if no $UNIT control card was given for the logical unit.) The message is printed as part of the output, and the job is terminated.

LU. xx, PU. xxx –
xxxx IS INVALID CONTROL MNEMONIC

A control operation has been requested, and the parameter list specified xxxx (an invalid code) as the mnemonic indicating the type of control to be performed. The message is printed as part of the output, and the job is terminated.

LU. xx, PU. xxx –
BSP (or FSP) REQ ON BUFFERED UNIT

A request has been made for either a backspace-record operation or a forward-space-record operation, but the $UNIT control card for this unit has specified that buffers should be used. The message is printed as part of the output, and the job is terminated.

LU. xx, PU. xxx – FILE PROTECTED

A write operation is requested for logical unit xxx, but the $REEL control card for this unit specified that the tape should be file-protected. The request is ignored, the message is printed as part of the output, and the job is terminated.

LU. xx, PU. xxx –
ILLEGAL DEVICE TYPE FOR SYSPRS (or SYSPUN or SYSRDS or SYSWRS)

An I/O operation has been requested which cannot be performed by the device selected.

LU. xx, PU. xxx –
ILLEGAL DEVICE TYPE FOR UNIT CONTROL

A tape-control operation has been requested, but the device specified in the parameter list is not a tape. The request is ignored, the message is printed as part of the output, a storage dump is taken, and the job is terminated.

LU. xx, PU. xxx –
INVALID DELIVERY AREA FOR READ

A read operation has been requested for unit xxx, but the parameter list specified either a register number greater than 13 or a storage address in an area of storage occupied by the monitor. The request is ignored, and the job is terminated.

LU. xx, PU. xxx –
LRS (or PRS) IN VAR BLKED REC
GREATER THAN BUF SIZE

A read-tape operation has been requested, but the logical record (or physical record) size of a variable-blocked record proves to be greater than the buffer area. The message is printed as part of the output, a storage dump is taken, and the job is terminated.

LU. xx, PU. xxx – RCD LNG LESS THAN 16

A write tape operation has been requested for unit xxx, but this request would cause IOPACK to create a noise record, i.e., a tape record of less than 16 bytes. The request is ignored, the message is printed as part of the output, and the job is terminated.

LU. xx, PU. xxx –
RCD SIZE GREATER THAN BUF SIZE

A write operation has been requested, but an error has been made in the parameter-list length specification. The message is printed as part of the output, a storage dump is taken, and the job is terminated.

* **MOUNT xxxxxx ON yyyyy – SYSTEM WAITING**

Indicated library, Compool, or SPT tape, specified on a $ENV card was not mounted, and the operator has been asked to change tapes.

* **NONEX ASSUMED**

A $XEQ, $AUX, or $NONEX control card was not found. The system assumes a $NONEX card with no units deleted.

OPERATOR OVERRIDE

The operator has continued the job without complying with the "MOUNT" message printed immediately above. Presumably, he had reason to believe that the $ENV card specified an incorrect tape label.

OPERATOR REQUESTED REST xxxxxx

or

OPERATOR REQUESTED SKIP xxxxxx

The operator has requested restart of a job (or skip to a job), where xxxxxx is the name of the requested job as given in columns 10–15 of the $ID control card. This message is printed to show why the current job was terminated with incomplete output.

* **REQUIRE MOD=SYST**

* **REQUIRE MOD=USER**

* **REQUIRE SES=a**

* **REQUIRE SYS=bbbbbb**

Any of these messages are printed to identify the $ENV card request which, unmet, forced job termination.

* SYSDUMP

A dump of main storage has been initiated as a result of an error or operator message.

XEQ(LIB is xxxxxx)

This message indicates that loading is complete, execution is starting, and the library tape labeled xxxxxx was used for any library routines required for the job.

JOB TERMINATED

This message indicates that an error was found that caused the current job to be terminated.

BATCH TERMINATED

This message indicates that an error was found which caused an entire batch of jobs to be terminated.

2.10 OVERRIDING SYSTEM PROTECTION FEATURES (USERIO)

Under three conditions, a user may need capability normally prohibited by the utility monitor:

1. To read patch cards for system processors.
2. To be permitted monitor mode during problem program execution.
3. To provide nonstandard I/O services to any type of device.

For any of these conditions, the user may specify "USERIO" when submitting the job.

Patching system processors is discussed in paragraph 1.3.4.

To be permitted monitor mode during program execution, the programmer should specify "USER" on the $ENV card or specify "USERIO" to the operator and include the following BAL sequence in the program:

```
SVC     SYSIOI

NI      X'68A', X'FE'
```

This will change the PSW to supervisor state with protect key of zero and I/O and external interrupts enabled. Register contents after SYSIOI are unpredictable. The NI instruction changes CRSL7 to indicate IOPACK is not operating.

2.10.1 Nonstandard I/O Services

The standard I/O routines in the 9020 Utility System for tape, card reader, card punch, printer, and console typewriter devices. The user may, however, wish to add nonstandard routines for these devices or I/O routines for other devices.

Any user I/O routine must exist within the framework of the present 9020 I/O and interrupt control system. Before writing his own I/O routine, the user must become familiar with the standard method of preparing I/O routines, and with the provisions offered by the utility system monitor.

2.10.2 Organization of I/O and Interrupt-Control System

Figure 2–14 diagrams the I/O and interrupt control system included in the utility system monitor. IOPACK is a set of routines that controls overlap and blocking, validates parameter lists, and directs operations. Interrupt control handles interrupts, including all that result from I/O and SVC instructions. The IOPACK routine transfers control to MONIO, which initiates an I/O operation on the specified device. For his own I/O routine, the user must write an IOPACK routine, the MONIO channel commands and status-testing code (hardware initiating routines), and interrupt routines.

The system can be understood best through an example. The problem program requests an I/O operation by a call (BAL with parameter list) to IOPACK. IOPACK puts itself in supervisor mode by executing an SVC SYSIOI; after validating linkages, it tests the LISTIO for the requested device to determine if the device is free. When the device is free, IOPACK moves the address of the current LISTIO to location CRCHPT (in the communications region), sets request flags in the LISTIO, and executes an SVC SYSIO to call MONIO.

Note: It is imperative that the user disable I/O interrupts before setting a pointer in CRCHPT and before setting request flags in the LISTIO.

Next, interrupt control, responding to the SVC SYSIO, exits to the MONIO initializing routine, which fetches the device type from the LISTIO and looks it up in the table MNLIST. For each device type, MNLIST has the address of two routines. The first is the starting address of the routine

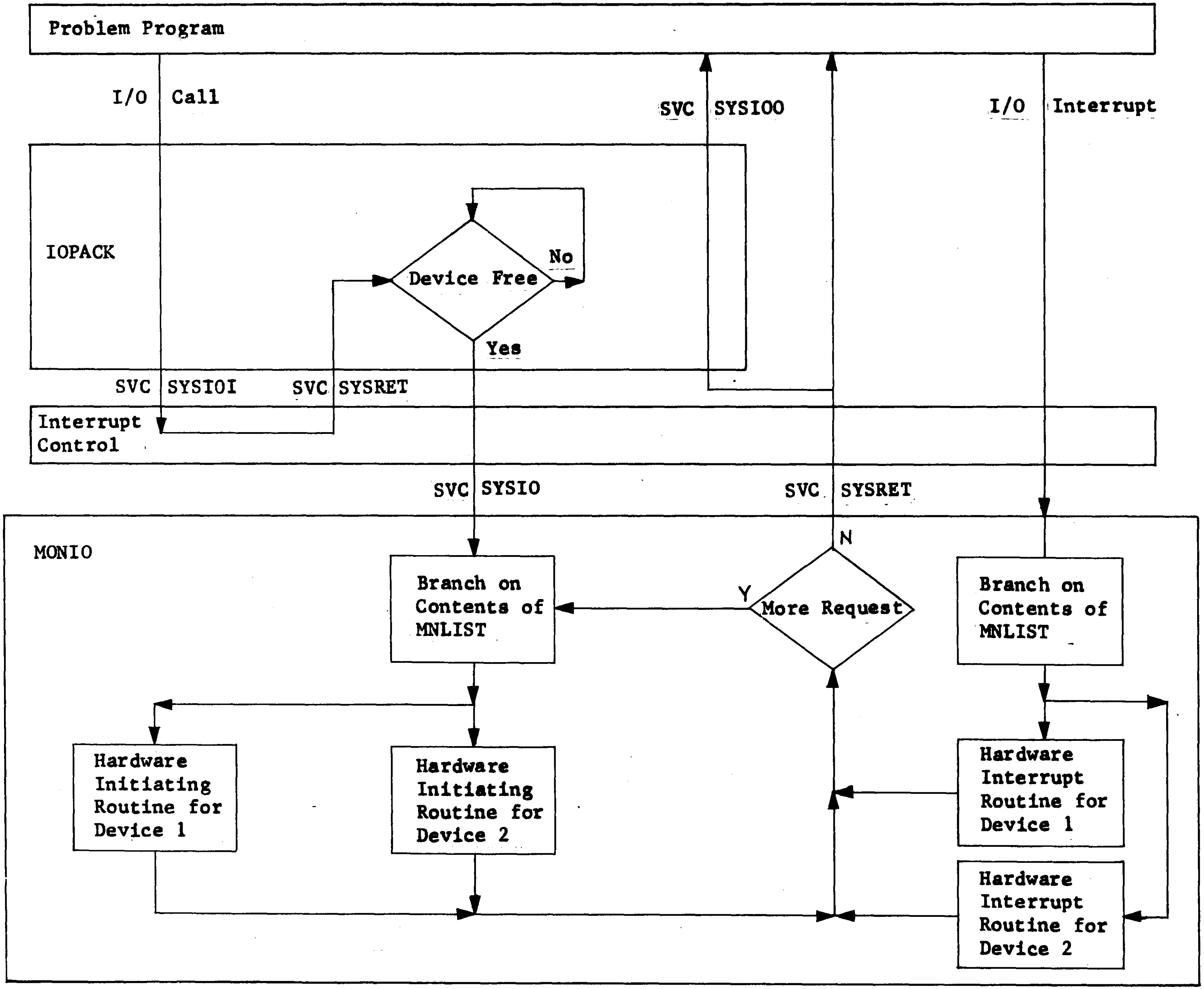

Problem Program
I/O Call
SVC SYSIOO
I/O Interrupt
IOPACK
Device Free
No
Yes
SVC SYSIOI
SVC SYSRET
Interrupt Control
SVC SYSIO
SVC SYSRET
MONIO
N
Y
More Request
Branch on Contents of MNLIST
Branch on Contents of MNLIST
Hardware Initiating Routine for Device 1
Hardware Initiating Routine for Device 2
Hardware Interrupt Routine for Device 1
Hardware Interrupt Routine for Device 2

Figure 2–14. I/O and Interrupt Control System

that handles interrupts for the device type; the second is the starting address of the routine that initiates the device. At this time, the initiating address is taken. The hardware initiating routine then tries to execute the necessary I/O instructions. If the initiating routine finds the channel or device busy, it returns control without initiating, and the request flag in the LISTIO remains on. After return to the initializing routine, the LISTIO's are searched for any requests that can be initated. After attempting to initiate any requests found, MONIO returns control to IOPACK via interrupt control. At this point, IOPACK executes an SVC SYSIOO to restore the operating mode (the first five bytes of the PSW) to its original state and returns to the problem program.

On an I/O interrupt, interrupt control calls the MONIO initializing routine to determine which channel interrupted and to search that channel's LISTIO chain for the corresponding device address. Next, according to the device type, MONIO branches to the interrupt address supplied in MNLIST to service the interrupt. If no device address match is found at this time, the interrupt is ignored. In either case, MONIO scans all other LISTIO's for the same channel and attempts to initiate any for which the request flag is set.

2.10.2.1 Standard IOPACK Routine

The standard IOPACK routine first executes an SVC SYSIOI to enter the supervisor mode, enabled for interrupts. Next, the routine does its blocking, deblocking, and related manipulation of logical records. If it is necessary to perform an I/O operation, the routine must test the LISTIO of the desired device to determine if the free flag is set to one and the request flag is set to zero. When this condition is met, IOPACK disables interrupts, sets the request flag to one, and places in CRCHPT the address constant that points to the current LISTIO. Finally, IOPACK executes an SVC SYSIO, transferring control to the MONIO intializing routine and from there to the hardware initiating routine. The IOPACK routine can pass parameters to the initiating routine via the LISTIO. See Figure 2–15.

2.10.2.2 Standard Hardware Routines

The hardware initiating routine starts the device; while the hardware interrupt routine handles interrupts (returning control to the initiating routine for error correction when appropriate). Both routines communicate with IOPACK through the LISTIO and CRCHPT. Errors may be corrected and CCW's constructed by either the hardware initiate routine or IOPACK.

For convenience, the MONIO initializing routine housekeeps general registers 5 through 12 as follows:

5 – Device address

6 – Direct exit (to MNXET when channel is busy)

7 – Scan exit (to MNXOT when channel is free)

8 – Kill-batch exit (to MNKL)

9 – Not used

10 – Interrupt routine address

11 – Initiate routine address

12 – Current LISTIO address

Register housekeeping is provided by the initializing routine and by MNXET and MNXOT. The console typewriter may be used freely, but registers 5–12 must be restored before exit. See Figure 2–16.

2.10.3 Utility Monitor Provisions – User Input/Output Routines

The monitor provides an option on the $UNIT card and a special supervisor call that allow user I/O routines to exist within the 9020 System.

By punching USERIO in place of IOPACK on the $UNIT control card, the user causes that card to be interpreted in a special way. The most important point is that a hardware device address is specified by the user. The LISTIO generator will not try to assign a device address but accepts the user-supplied address. (A check is made, however, to see that this address is not assigned to a basic system unit – .INPUT, .PRINT, .PUNCH, .AUXIL, or .SYSTM. If the address is assigned to a basic unit and USERIO request is for a problem program, the job will be terminated. System processor USERIO requests are also checked for identical device addresss. If the address duplicates that of an existing unit, a diagnostic is issued and the request is ignored.) Also, a flag is set in the communications region (in CRIOSW) to show that a user I/O routine has been requested.

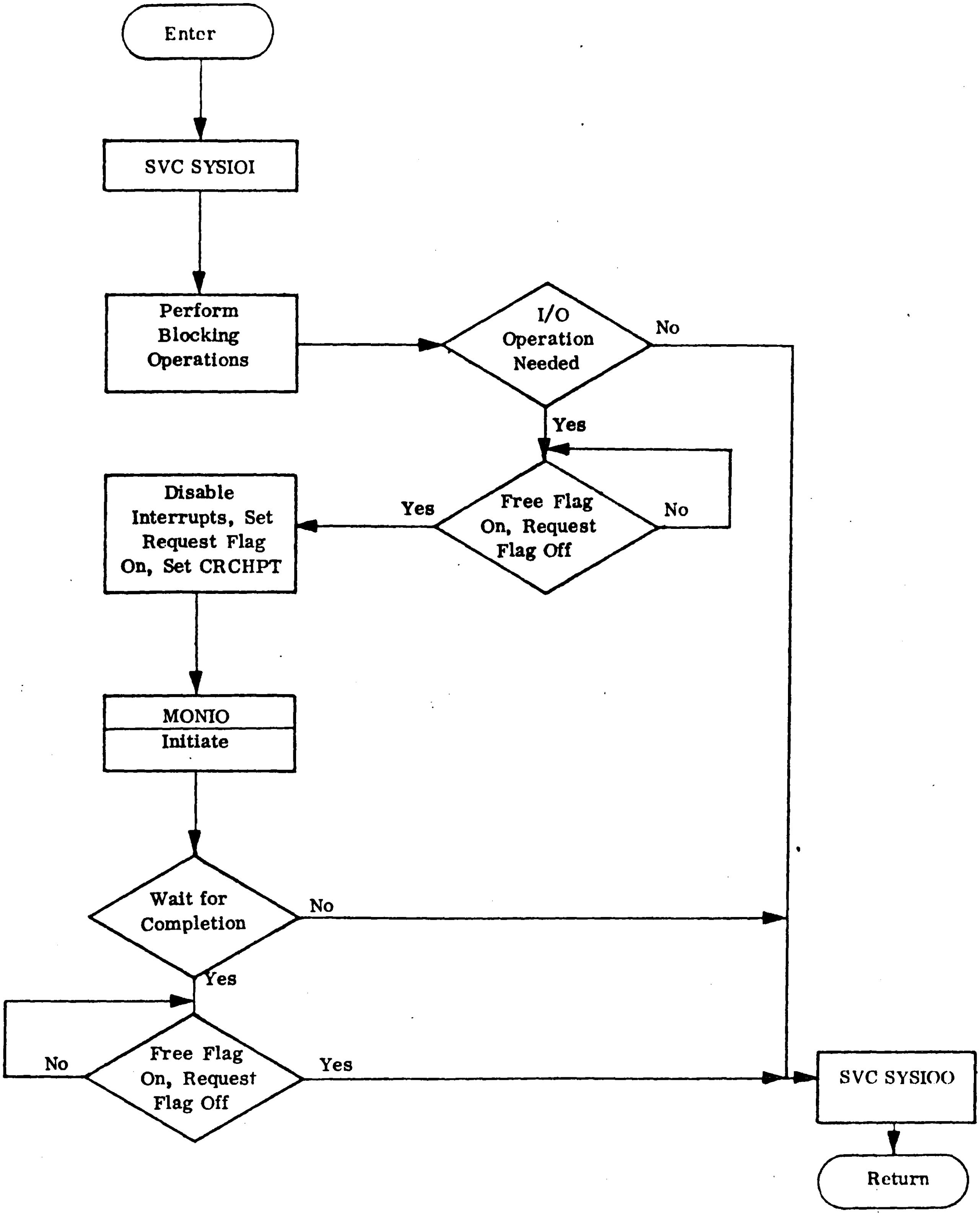

Enter
SVC SYSIOI
Perform Blocking Operations
I/O Operation Needed
No
Yes
Disable Interrupts, Set Request Flag On, Set CRCHPT
Yes
Free Flag On, Request Flag Off
No
MONIO
Initiate
Wait for Completion
No
Yes
No
Free Flag On, Request Flag Off
Yes
SVC SYSIOO
Return

Figure 2–15. IOPACK Routine

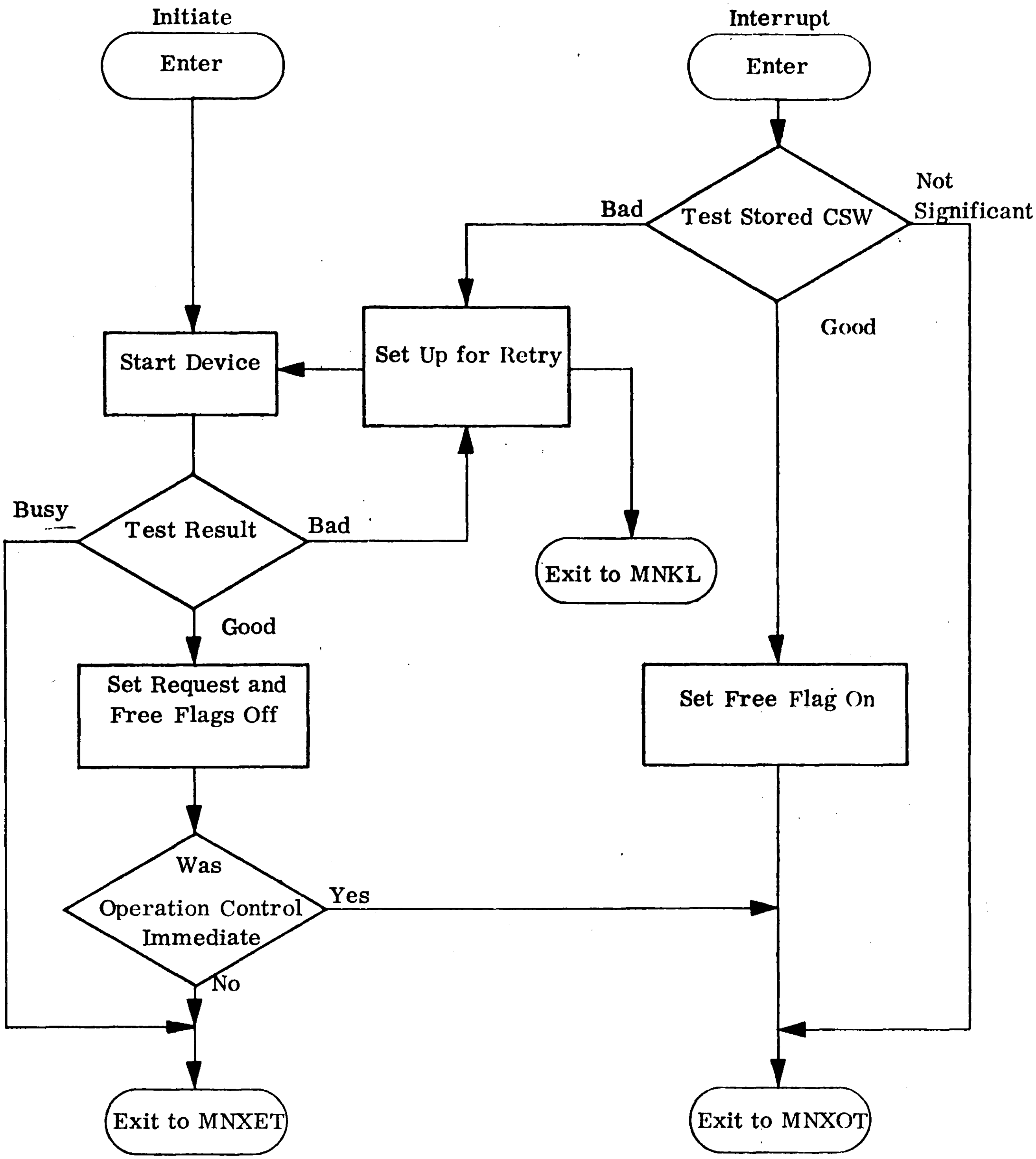


Figure 2–16. Hardware Initiate and Interrupt Routine

Before attempting to initiate an I/O device, the user must give the monitor the address of his I/O interrupt and hardware initiating routines. This is done by an SVC SYSPIR, as follows:

```
CNOP    2,4
SVC     SYSPIR
DC      A (interrupt)     for device type 06
DC      A (initiate)

DC      A (interrupt)     for device type 07
DC      A (initiate)

DC      A (interrupt)     for device type 08
DC      A (initiate)

DC      A (interrupt)     for device type 09
DC      A (initiate)

DC      A (interrupt)     for device type 10
DC      A (initiate)
```

All these entries must be given in the order listed even if some are unused.

The format of a USERIO $UNIT control card is explained in paragraphs 2.2.5.1.2 and 2.2.5.1.3.

The device type field is used by the MONIO cover routine to determine which interrupt/initiating routine is to be called. It is recommended that similar devices be given the same device type.

For problem programs, USERIO cards must precede IOPACK $UNIT cards so that physical units specified on these cards will not be assigned to logical units specified on the latter cards.

2.10.3.1 Precautions

The user I/O routine becomes, in effect, a part of the utility monitor and can, if in error, destroy the monitor. The writer of the user I/O routine must guard against errors in his I/O routine, as well as bad linkages and other errors from the calling program; also, he must make certain that his devices are dormant at end of job.

The LISTIO constructed for a user device has only the physical device address, the device type, the channel pointer, the request flag, the free flag, and the critical unit flag (for example, .SYSTM). The channel address word (CAW) is also set for a system processor USERIO request. The user can employ the other LISTIO fields as desired.

The IBM 9020 Utility System I/O support (i.e., IOPACK and MONIO) is not recursive. Therefore, the system mask may not be set to 1 bits while an I/O routine is operating. (This restriction does not apply to IOPACK routines until they set flags in the LISTIO or set a point in CRCHPT. The console typewriter, however, may be used freely.)

No dynamic dumps may be taken in I/O routines.

2.10.3.2 Suggestions

Typical structure of a user hardware initiating routine is shown in Figure 2–17.

The user can obtain the address of a particular LISTIO by executing this linkage to the LIOGET routine:

```
L       15, CRLGET
BALR    14,15
DC      H'logical unit'
```

where logical unit is the logical unit number. When this is executed, the LISTIO address will be placed in register 13. If the designated logical unit has not been assigned (i.e., it has no LISTIO), register 13 will be set to zero. The user routine must be in the Supervisor mode (by executing an SVC SYSIOI) before this linkage is executed.

The user can find the addresses of all communications region locations from the utility monitor listing. These addresses are fixed and can be equated to absolute expressions when interfacing with the utility system monitor.

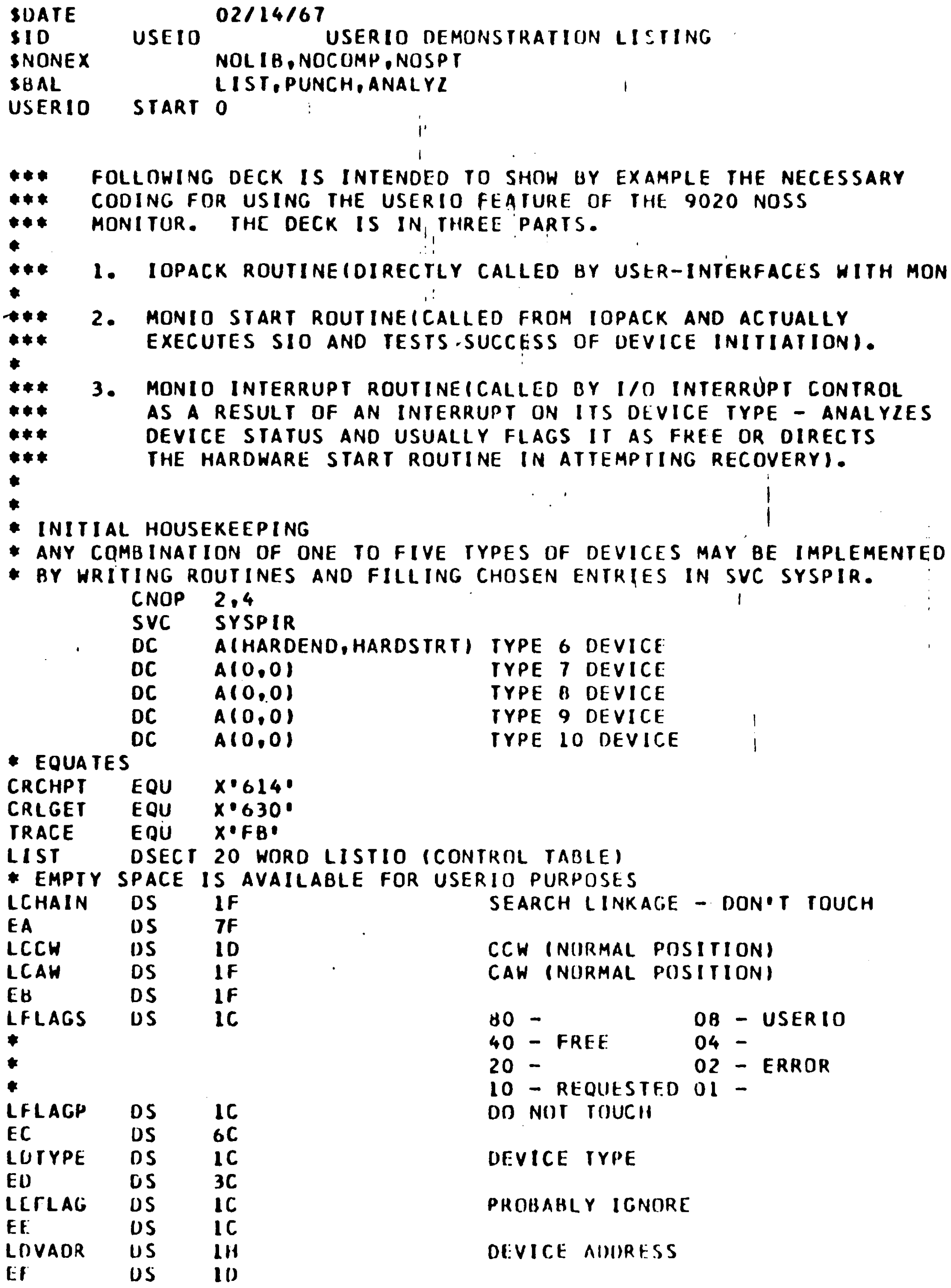
2.11 BATCH

The BATCH routine is a monitor facility which allows for several off-line operations to be handled simultaneously in a small machine configuration. A maximum of three operations requiring print/punch groups can be done at one time. Simultaneous operations include:

a. Create a utility system format SYSIN tape with either cards, or cards and an SPT tape as input.

b. Create an unblocked card image tape.

c. Print tapes in SYSOUT, SYSIN, unblocked print record, hexadecimal dump, or alphabetic dump format.



```
$DATE            02/14/67
$ID       USEIO          USERIO DEMONSTRATION LISTING
$NONEX           NOLIB,NOCOMP,NOSPT
$BAL             LIST,PUNCH,ANALYZ
USERIO    START  0


***    FOLLOWING DECK IS INTENDED TO SHOW BY EXAMPLE THE NECESSARY
***    CODING FOR USING THE USERIO FEATURE OF THE 9020 NOSS
***    MONITOR.  THE DECK IS IN THREE PARTS.
*
***    1.  IOPACK ROUTINE(DIRECTLY CALLED BY USER-INTERFACES WITH MONIO)
*
***    2.  MONIO START ROUTINE(CALLED FROM IOPACK AND ACTUALLY
***        EXECUTES SIO AND TESTS SUCCESS OF DEVICE INITIATION).
*
***    3.  MONIO INTERRUPT ROUTINE(CALLED BY I/O INTERRUPT CONTROL
***        AS A RESULT OF AN INTERRUPT ON ITS DEVICE TYPE - ANALYZES
***        DEVICE STATUS AND USUALLY FLAGS IT AS FREE OR DIRECTS
***        THE HARDWARE START ROUTINE IN ATTEMPTING RECOVERY).
*
*
* INITIAL HOUSEKEEPING
* ANY COMBINATION OF ONE TO FIVE TYPES OF DEVICES MAY BE IMPLEMENTED
* BY WRITING ROUTINES AND FILLING CHOSEN ENTRIES IN SVC SYSPIR.
          CNOP   2,4
          SVC    SYSPIR
          DC     A(HARDEND,HARDSTRT) TYPE 6 DEVICE
          DC     A(0,0)              TYPE 7 DEVICE
          DC     A(0,0)              TYPE 8 DEVICE
          DC     A(0,0)              TYPE 9 DEVICE
          DC     A(0,0)              TYPE 10 DEVICE
* EQUATES
CRCHPT    EQU    X'614'
CRLGET    EQU    X'630'
TRACE     EQU    X'FB'
LIST      DSECT 20 WORD LISTIO (CONTROL TABLE)
* EMPTY SPACE IS AVAILABLE FOR USERIO PURPOSES
LCHAIN    DS     1F                  SEARCH LINKAGE - DON'T TOUCH
EA        DS     7F
LCCW      DS     1D                  CCW (NORMAL POSITION)
LCAW      DS     1F                  CAW (NORMAL POSITION)
EB        DS     1F
LFLAGS    DS     1C                  80 -            08 - USERIO
*                                    40 - FREE       04 -
*                                    20 -            02 - ERROR
*                                    10 - REQUESTED  01 -
LFLAGP    DS     1C                  DO NOT TOUCH
EC        DS     6C
LDTYPE    DS     1C                  DEVICE TYPE
ED        DS     3C
LCFLAG    DS     1C                  PROBABLY IGNORE
EE        DS     1C
LDVADR    DS     1H                  DEVICE ADDRESS
EF        DS     1D
```

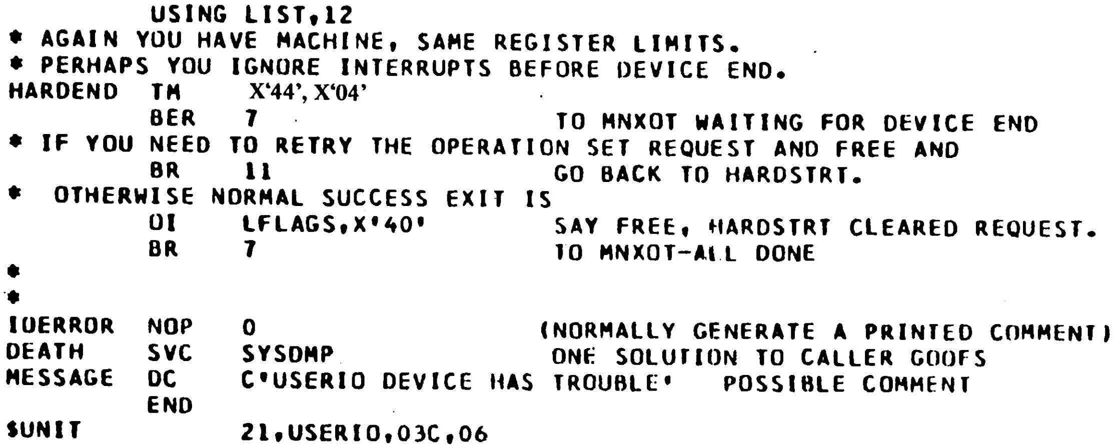
Figure 2–17. Typical User IOPACK and Hardware Routines (Sheet 1 of 4)

```
LEOFAD   DS    A(0)                EOF RETURN (SEE SYSSR)
LERRAD   DS    A(0)                ERR RETURN (SEE SYSSR)
PROG     CSECT
*
*
*
**********************************************************************
* IOPACK (LOGICAL IOCS)
* NORMALLY WRITTEN AS A SUBROUTINE, THOUGH NOT NECESSARILY.
* ALWAYS PAIR SYSIOI AND SYSIOO
* PRESUMED CALL IN EXAMPLE IS
*        L     15,=A(SOFTSTART)
*        CNOP  2,4
*        BALR  14,15
*        DC    A(ARG)
ZZ       DSECT                     TO DESCRIBE ARGUMENT AREA
ARG      DS    H                   LOGICAL UNIT
PROG     CSECT
* SUGGESTED CODE WILL REQUEST MONITOR MODE, PROMISE SYSIOO, AND
*   SAVE REGISTERS.
SOFTSTRT SVC   SYSIOI
         BALR  12,0                SYSIOI DESTROYED 15-ANYWAY 12 IS STANDARD
         USING *,12
         L     11,0(0,8)           8 HAS COPY OF 14
         USING ARG,11              14 ALSO WAS LOST IN SYSIOI
         MVC   X,ARG               LOGICAL UNIT FOR CRLGET
         L     15,CRLGET
         BALR  14,15
X        DC    H'0'                LOGICAL UNIT
         LTR   13,13               WAS THERE A LOGICAL UNIT
         BZ    DEATH               NO-CALLER GOOFED
         USING LIST,13             CAN REFERENCE LISTIO
         CLI   LDTYPE,6
         BNE   DEATH               LU HAS WRONG DEVICE TYPE
***      DETERMINE ADDRESS AND LENGTH OF DATA OR REQUIRED CONTROL COMMAND.
***      IF NONE REQUIRED(BLOCKING, ETC.) GOTO LPC.
LPA      TM    LFLAGS,X'10'        IS REQUEST FLAG ON
         BNE   LPA                 WAIT FOR PREVIOUS BUFFERED OPERATION
         TM    LFLAGS,X'40'        IS PREVIOUS OPERATION FINISHED
         BZ    LPA                 NO-WAIT
*        USE FOLLOWING ONLY IF YOUR MONIO ROUTINE SUPPORTS IT.
         TM    LFLAGS,X'02'
         BNE   IOERROR             (DETECTED BY YOUR MONIO ROUTINE)
***      CONSTRUCT CCW'S IN AREA ASSIGNED THIS DEVICE(LCCW).
***      CONSTRUCT CAW IN AREA ASSIGNED THIS DEVICE (LCAW).
         SSM   =X'0000'            MUST INHIBIT INTERRUPTS
         ST    13,CRCHPT           SET POINTER FOR MONIO
         OI    LFLAGS,X'10'        INDICATE NEED FOR SIO
         SVC   SYSIO               CALL MONIO
         SSM   =X'FFF0'            ALWAYS RE-ENABLE INTERRUPTS
***      GOTO LPB OR LPC AS APPROPRIATE (OR EVEN LPA).
LPB      TM    LFLAGS,X'10'        FOR UNBUFFERED OPERATION IOPACK
         BNE   LPB                   WAITS FOR SUCCESSFUL DEVICE END
         TM    LFLAGS,X'40'
         BZ    LPB
```

Figure 2–17. Typical User IOPACK and Hardware Routines (Sheet 2 of 4)

```
***      IF YOU WAITED YOU CAN TEST FOR SUCCESS.
LPC      SVC   SYSIOO              PAIRS WITH SYSIOI/RESTORES ORIGINAL
*                                    REGISTERS
         B     4(0,14)             RETURN TO CALLER
         DROP  11,12,13
         LTORG
*
*
*
********************************************************************
* MONIO (PHYSICAL IOCS)
*
**** WARNING-HARDEND GETS CALLED WHENEVER DEVICE INTERRUPTS-EVEN IF
*    NOT SELECTED.  HARDSTRT GETS CALLED IF REQUEST FLAG IS ON AND THERE
*    IS ACTIVITY ON THE CHANNEL AND DEVICE IS FREE.  THEREFORE YOU ARE
*    MULTIPROGRAMMING -1- LEVEL AND SHOULD NOT CALL ANY COMMON OR LIBRARY
*    ROUTINES FROM HARDSTRT OR HARDEND.  THIS IS WHY THE LISTIO
*    HAS COMMUNICATION FIELDS.
*
* REGISTERS 0-4 ARE AVAILABLE - 5-12 MUST BE RESTORED IF CHANGED.
         USING HARDSTRT,11         SET FOR YOU
         USING LIST,12
HARDSTRT TM    LEFLAG,X'20'        HAS YOUR INTERRUPT ROUTINE MARKED
*                'UNIT NOT READY-  IF SO WAIT FOR DEVICE END INTERRUPT.
         BNER  7                   GIVE CONTROL TO MNXOT
         OI    LFLAGS,X'50'        SET FREE AND REQUEST
         MVC   72(4),LCAW          (IF YOU KEPT CAW IN LISTIO)
*  HA IS NORMAL FOR DATA TRANSFER COMMANDS, HB IS FOR CONTROL.
HA       SVC   TRACE               MAINTAIN HISTORY FOR DEBUGGING
         SIO   0(5)                PRECEDE ALL I/O INSTRUCTIONS WITH TRACE
         BNE   TRBL                DATA COMMAND SHOULD GIVE CC=0
         NI    LFLAGS,X'AF'        SAY NOT REQUESTED AND NOT FREE (GOOD)
         NI    LEFLAG,X'BF'        SAY BUSY
         BR    6                   TO MNXET (DONE)
*TRBL* PLAY THIS BY EAR, YOU HAVE INTERRUPTS DISABLED AND CAN
*    USE SIO, TIO, ETC, ALWAYS PRECEDED BY SVC TRACE
*    1052 IS AVAILABLE (SYSTWR, SYSTRE) BUT NO OTHER I/O
TRBL     SVC   SYSTWR
         DC    AL3(MESSAGE)
         DC    AL1(L'MESSAGE)
* FOR SERIOUS TROUBLE, BATCH CAN BE KILLED WITH ON-LINE EMERGENCY DUMP,
         BR    8                   TO MNKL
*        OR INTERRUPTS MAY BE PERMITTED BY RETURNING TO IOPACK ON BUSY COND.
         BR    7                   TO MNXOT
*CONTROL IS SIMILAR BUT CSW IS EXPECTED TO BE STORED ON SUCCESS.
HB       NI    LFLAGS,X'AF'        NOT REQUESTED AND NOT FREE
         TM    68,X'04'            LOOK FOR DEVICE END
         BER   6                   MNXET IF DEVICE NOT DONE
         MVI   LEFLAG,X'40'        SAY NOT BUSY
         OI    LFLAGS,X'40'        SAY FREE-ALL DONE
         BR    7                   TO MNXOT TO  SCAN REST OF CHANNEL
         DROP  11,12
*MONIO INTERRUPT ROUTINE*****
*
         USING HARDEND,10          SET FOR YOU
```

Figure 2–17. Typical User IOPACK and Hardware Routines (Sheet 3 of 4)



```
               USING LIST,12
* AGAIN YOU HAVE MACHINE, SAME REGISTER LIMITS.
* PERHAPS YOU IGNORE INTERRUPTS BEFORE DEVICE END.
HARDEND  TM    X'44',X'04'
         BER   7                   TO MNXOT WAITING FOR DEVICE END
* IF YOU NEED TO RETRY THE OPERATION SET REQUEST AND FREE AND
         BR    11                  GO BACK TO HARDSTRT.
*  OTHERWISE NORMAL SUCCESS EXIT IS
         OI    LFLAGS,X'40'        SAY FREE, HARDSTRT CLEARED REQUEST.
         BR    7                   TO MNXOT-ALL DONE
*
*
IOERROR  NOP   0                   (NORMALLY GENERATE A PRINTED COMMENT)
DEATH    SVC   SYSDMP              ONE SOLUTION TO CALLER GOOFS
MESSAGE  DC    C'USERIO DEVICE HAS TROUBLE'   POSSIBLE COMMENT
         END
$UNIT          21,USERIO,03C,06
```

Figure 2–17. Typical User IOPACK and Hardware Routines (Sheet 4 of 4)

d. Log any tape.

e. Punch a SYSIN tape, an AUXIL tape, a hexadecimal tape, or an unblocked card image tape.

f. Write end-of-file tape marks on tape.

Other off-line operations performed by BATCH, but which lock out all other operations are:

a. Create a merged library and compool tape.

b. Duplicate one tape onto another one.

c. Compare two tapes in SYSOUT, SYSIN, SPT, or hexadecimal tape format.

d. Create a 7-track SYSOUT tape from a 9-track SYSOUT.

The use of BATCH in an off-line and input/output system set up is described in Section 3, System Tape Generation. If a hard copy of a deck which is only resident on tape is wanted, then the punching of a .SYSIN or .AUXIL tape can be very useful.

The duplication of tape from 9-track to 7-track or just a straight duplication can be used for tape backup, or to make use of the 1401, or 7-track drives on the 9020.

Except for the generated SYSIN tape, which is always written on .AUXIL, all tape assignments are made by operator messages assigning specific drives to specific functions. Every tape drive listed in the IPL time MOUNT message, and normally any other attached tape, is available for any BATCH function (except as noted under individual functions).

Control of BATCH is via typewriter input messages. These must be entered in response to one of the three following control messages:

1. INITIALIZE BATCH – The processor has just been loaded via the IPL message BATCH.

2. GO! – BATCH awaits a new set of control messages in response to the typewriter sequence: depress REQUEST key, wait for ENTER REQUEST, depress ENTER key.

3. TRY AGAIN STARTING WITH xxxxx——BATCH rejected the first message indicated (xxxxx) but has not yet decoded any following submessages. All of these remain in the message area, and what is typed at this time will replace a corresponding portion of the original messages. It is not necessary to retype the entire original message; change only the portion that was incorrect. (If new submessages are shorter, blanks may be left at the end of any submessage. However, if they are longer the entire remainder of the message must be retyped.)

2.11.1 Create SYSIN Tapes

To initiate SYSIN tape generation, stack job decks in the reader and type in CTT. For an unblocked card image SYSIN, stack job decks in the reader and type CTT/N. In final job processing SYSIN will be on tape. Use $ENDTAPE instead of $END to terminate the last batch to be loaded on any SYSIN reel. (Decks for more than one reel may be stacked.)

BATCH scans all cards written on the generated SYSIN and types a summary consisting of significant portions of each $ID, $UNIT, $REEL, and $ENV control card.

BATCH enables the programmer to submit any assortment of decks (or selected portions of those decks) prestored on one or more SPT's with insert, delete, merge, or replace cards on SYSIN. To use this service, proceed as follows:

a. Place selected card decks on an SPT tape. BATCH imposes only two restrictions:

    1. The tape label must be unique.

    2. Each card image on the SPT tape is assumed to contain an 8-character sequence number in columns 73–80, and cards in each file must be in order if updates are to be used.

    Assuming that the submitted decks are acceptable to SPTEDT, BATCH may process any of these decks.

b. Assign a unit(s) for the SPT tape(s). To assign a drive for SPT, enter the message SPTnnn. More than one drive may be assigned by typing in several SPT messages with different unit numbers. The units will be used in the order assigned as requests for new tapes are processed.

c. Prepare job deck. To control merging of cards from SPT tapes, include $SPT control cards in the following formats as appropriate:

Column	**1**	**1**
1	**0**	**6**
$SPT	zzzzzz	xxxxxx
$SPT	zzzzzz	xxxxxx, yyyyyyyy
$SPT	zzzzzz	******, yyyyyyyy
$SPT	zzzzzz	

where:

xxxxxx is SPT tape label from the $SPTEDT card.

yyyyyyyy is SPT file name from the APROG card.

zzzzzz is an optional SYSIN resequencing specification. The field may be blank, or may contain RESEQ, NORES, or ABSNR.

The first format, specifying only a tape label, serves to select the specified tape (normally for a subsequent $SPT card specifying file name but not tape label), is considered a $ control card, and serves no other purpose. It may appear any place in a job deck prior to other $SPT control cards for the same tape.

The second format, specifying tape label and file name, serves to select the specified tape and initiate processing of the named file. It is self-sufficient and does not require a preceding $SPT tape select card.

The third format, containing******in the tape select field, serves to initiate processing of the named file of the currently selected tape. If one or more $SPT cards of this format are used, they must be preceded by a $SPT card specifying tape label.

The fourth format is used where a $ control card is required only to satisfy the BATCH rule that processing of an SPT file may be terminated only by a $ control card or 7/8EOF card.

Follow the $SPT file select card (second or third format) with INS/FINIS, DEL, merge, and replace cards in the same manner as for an SPT update (except that BATCH will accept any valid sequence number without demanding an identical number from tape for INS and DEL cards). Terminate processing the file with whatever $ control card is appropriate. (Note that the last form of $SPT card is provided for the special case where no $ control card is desired in the generated SYSIN deck and no other $SPT card is appropriate. Also note that, for the corresponding case where a $ control card is to be inserted without terminating file processing, the user must place his card (with other cards as appropriate) between INS and FINIS cards and place a pseudo-sequence number of $$$$$$$$ on the $ control card.)

For card to tape errors using an SPT tape, some of the messages produced are:

```
$SPT****FILExxxxxxyyyyyyyy: DELaaaaaaaa,        bbbbbbbbAFTERcccccccc
$SPT****FILExxxxxxyyyyyyyy: INS  aaaaaaaa        AFTERcccccccc
$SPT****FILExxxxxxyyyyyyyy: MERGE/REPLACEbbbbbbbbAFTERcccccccc
$SPT****FILExxxxxxyyyyyyyy: NO FINIS FOR INSaaaaaa
$SPT****FILExxxxxxyyyyyyyy: SPT CARD HAS AN ERROR
$SPT****FILExxxxxxyyyyyyyy: RESEQUENCED SPT NUMBERS OVER 99999000
   MOUNT SPT –xxxxxx– ON DRIVE yyy (not aaaa bbbbbb).
```

For all but the last error message, looking at the deck should be enough to find the problem.

The last error message indicates either the wrong tape was mounted or the wrong tape was requested on the instructions to the operator. The operator may respond to this message in one of three ways:

1. Mount the correct SPT tape.
2. Type in 'CAN'. This cancels the SPT request.
3. Type in 'IGN' and load the SPT down. This causes BATCH to ignore the tape label and use the SPT that is mounted.

BATCH can resequence the card images placed on SYSIN tape. Resequencing is identical to SPTEDT resequencing. This function permits knowledge of the new sequence number prior to performing the actual SPT edit.

Columns 10–14 of the $SPT card are used to designate the type of resequencing desired. When columns 10–14 are blank or contain ABSNR, BATCH performs no resequencing.

When columns 10–14 of the $SPT card contain RESEQ, the cards are resequenced by 1000. The source change cards are applied to the SPT file, inserting, deleting, or replacing as necessary. Each record is then assigned a new sequence number which is 1000 greater than the last number. The record with the new number is then written on the SYSIN tape.

When columns 10–14 of the $SPT card contain NORES, only those cards appearing within an INS/FINIS sequence and containing blanks in columns 73–80 are assigned a sequence number. A card within an INS/FINIS is considered blank if a number which is present in the card is lower than the last assigned number. A number which is contained in the card and is in higher collating sequence is accepted and used by BATCH. Alphabetic characters are acceptable if they are in higher collating sequence.

When the NORES option is used, incrementing occurs by one. BATCH creates a new sequence number by using the last valid sequence number and updates starting at the least significant (one's) digit, generating carry updates only as far to the left as necessary. Any digits updated by BATCH in any character position contain only the Hollerith characters 1 through 9, never zero (0).

For example, an insert of 12 cards between sequence numbers 00005000 and 00006000, with all cards except the seventh containing blanks, produces the following results:

original card 00005000	**00005000**
insert card 1	**00005001**
insert card 2	**00005002**
insert card 3	**00005003**
insert card 4	**00005004**
insert card 5	**00005005**
insert card 6	**00005006**
insert card 7 (with number 00005997)	**00005997**
insert card 8	**00005998**
insert card 9	**00005999**
insert card 10	**00006111**
insert card 11	**00006112**
insert card 12	**00006113**
original card 00006000	**00006114**
original card 00007000	**00007000**

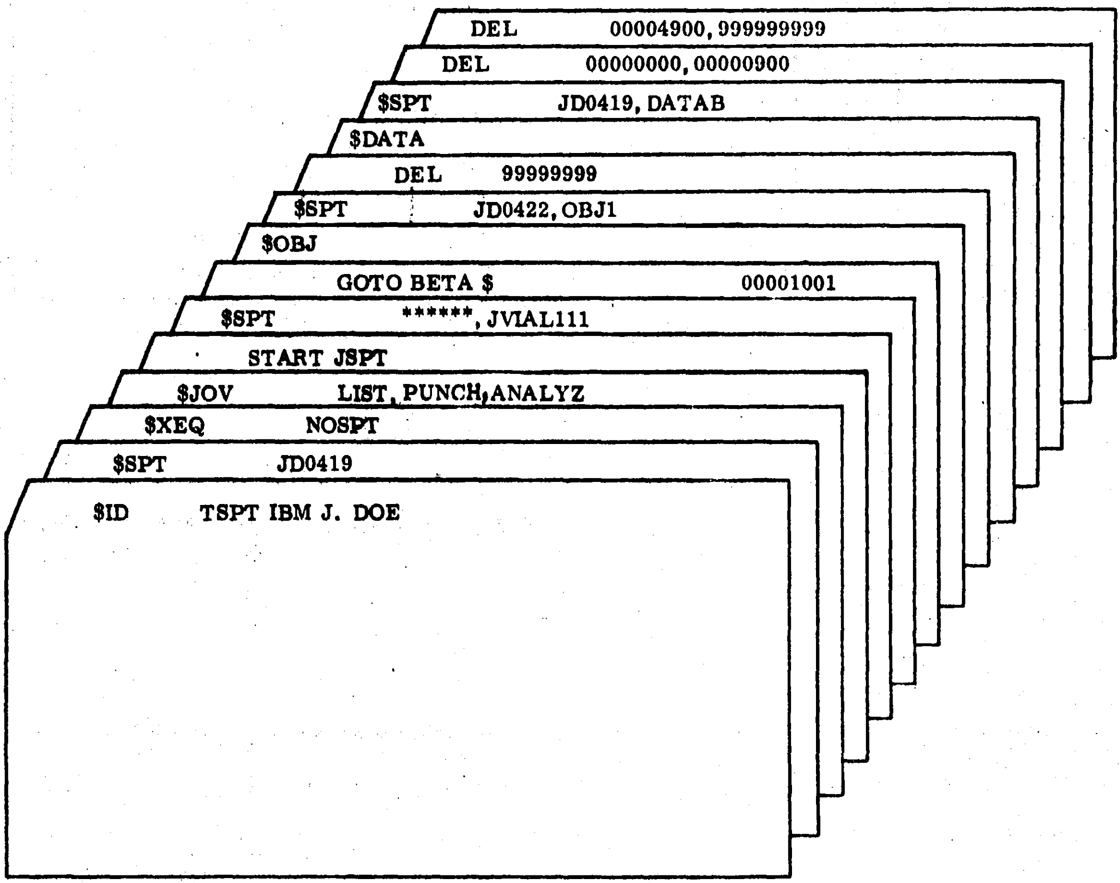
Note: It can easily be seen that card number 00007000 is an original card, since this mode never generates a zero.

BATCH interprets every $SPT card individually, and performs sequencing as indicated on that $SPT card. That type of sequencing continues until either a new $ID card or a new $SPT card is encountered.

The sample job deck (Figure 2–18) references three files on two SPT tapes. The first reference is to a JOVIAL source deck on tape JD0419. A START card was inserted before the $SPT card, presumably because there was no START card in the SPT file.

The second reference is to an object deck on tape JD0422. The DEL card is probably meaningless since card number 99999999 does not normally exist.

The third reference is to a data deck, again on tape JD0419. The control cards delete all except cards 00000901 through 00004899. These cards are then resequenced from 00002000 through 04000000. The $SPT card represents 00001000.



```
$ID        TSPT IBM J. DOE
$SPT            JD0419
$XEQ            NOSPT
$JOV            LIST, PUNCH,ANALYZ
     START JSPT
$SPT                ******, JVIAL111
            GOTO BETA $                              00001001
$OBJ
$SPT                JD0422, OBJ1
            DEL         99999999
$DATA
$SPT                JD0419, DATAB
            DEL         00000000, 00000900
            DEL         00004900, 999999999
```

Figure 2–18. Sample Job Deck

2.11.1.1 SYSIN Creation Using Conditional Copy

This facility is an extension of the basic SYSIN creation outlined in the preceding paragraphs. This capability provides a method for conditionally extracting data from an SPT file. The burden of duplicate maintenance is eased when all source data relating to a particular program resides on one SPT file. The differentiation between code applicable to versions of the same program will be accomplished by the use of control statements recognized only by BATCH. These control statements may exist either as card input or may actually reside on the SPT file itself. A user-invoked SPT option specifying the version(s) of code desired from the SPT file will permit the extraction of data from the tape.

This BATCH facility will permit, for example, a 3d1.0 version of a given program to be produced from a combined 3c/3d source SPT file by extracting as output (SYSIN) only the code which applies to the 3d version. Thus, code common to both versions appears only once in an SPT file.

BATCH will now recognize two new $SPT cards (MATCH and MATCX) as well as a new file alteration card (ALLOW).

The new $SPT cards have the following form:

Column	**1**	**10**	**16**
	$SPT	**MATCH**	**AA, BB, ...**
	$SPT	**MATCX**	**AA, BB, ...**

From 0 to 5 positional 2-byte parameters beginning in column 16 on the MATCH or MATCX card will be permitted by BATCH.

The new file alteration card has the following form:

Column	**10**	**16**
	ALLOW	**AA, BB, ...**

From 0 to 19 positional 2-byte parameters beginning in column 16 on the allow card will be permitted by BATCH.

The following rules will apply to the MATCH or MATCX option:

a. If no MATCH or MATCX option is detected, BATCH will operate exactly as it does now (i.e., all data on SPT transferred to SYSIN including ALLOW cards; if present).

b. An encountered MATCH or MATCX option will remain in effect until overridden by another MATCH or MATCX option or by a $ID card.

c. A MATCH or MATCX option without parameters (blanks in columns 16–29) accepts all source as eligible.

d. A MATCH or MATCX option will accept as eligible all source encountered prior to the first ALLOW card, per SPT file.

The following rules will apply to the ALLOW File Alteration card:

a. ALLOW cards will be stripped from output (SYSIN) if a MATCH option is in effect; however, all ALLOW cards will be transferred to SYSIN if a MATCX option is in effect.

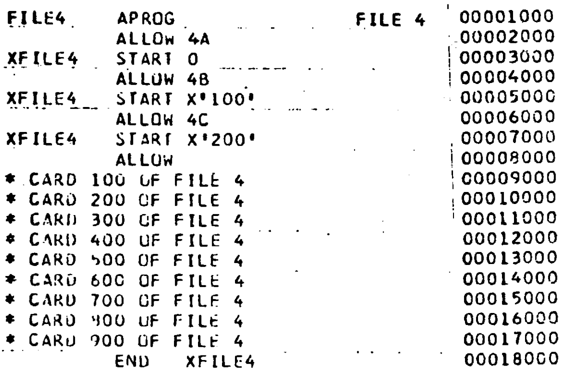
b. ALLOW arguments apply to all source following until another ALLOW card is encountered, per SPT file.

c. ALLOW cards with no arguments (blanks in columns 16–71) indicate that source following will always be included in output (SYSIN).

Following are examples of job setup and expected results:

Example 1

SPT Contents:

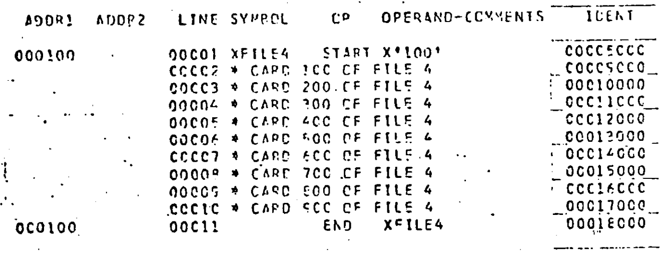


```
FILE4    APROG                    FILE 4   00001000
         ALLOW 4A                          00002000
XFILE4   START 0                           00003000
         ALLOW 4B                          00004000
XFILE4   START X'100'                      00005000
         ALLOW 4C                          00006000
XFILE4   START X'200'                      00007000
         ALLOW                             00008000
* CARD 100 OF FILE 4                       00009000
* CARD 200 OF FILE 4                       00010000
* CARD 300 OF FILE 4                       00011000
* CARD 400 OF FILE 4                       00012000
* CARD 500 OF FILE 4                       00013000
* CARD 600 OF FILE 4                       00014000
* CARD 700 OF FILE 4                       00015000
* CARD 800 OF FILE 4                       00016000
* CARD 900 OF FILE 4                       00017000
         END    XFILE4                     00018000
```

Control Card Input:

```
1          10          16

$ID        NAME
$BAL                   LIST, ANALYZ
$SPT       MATCH       4B
$SPT                   SPT002, FILE 4
```

Assembler Output:

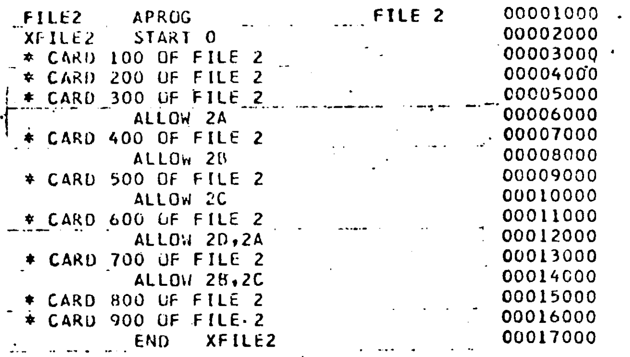


```
ADDR1  ADDR2   LINE SYMBOL     OP   OPERAND-COMMENTS      IDENT

000100         00001 XFILE4    START X'100'              00005000
               00002 * CARD 100 OF FILE 4                00005000
               00003 * CARD 200 OF FILE 4                00010000
               00004 * CARD 300 OF FILE 4                00011000
               00005 * CARD 400 OF FILE 4                00012000
               00006 * CARD 500 OF FILE 4                00013000
               00007 * CARD 600 OF FILE 4                00014000
               00008 * CARD 700 OF FILE 4                00015000
               00009 * CARD 800 OF FILE 4                00016000
               00010 * CARD 900 OF FILE 4                00017000
000100         00011           END   XFILE4              00018000
```

In example 1 the proper START card was selected by using the MATCH "4B" control card. The "ALLOW blank" card was used to include all source from that point to the end of file (or next ALLOW control card).

Example 2

SPT Contents:

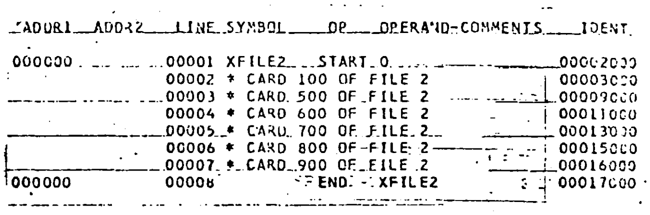


```
FILE2     APROG                 FILE 2     00001000
XFILE2    START 0                          00002000
* CARD 100 OF FILE 2                       00003000
* CARD 200 OF FILE 2                       00004000
* CARD 300 OF FILE 2                       00005000
          ALLOW 2A                         00006000
* CARD 400 OF FILE 2                       00007000
          ALLOW 2B                         00008000
* CARD 500 OF FILE 2                       00009000
          ALLOW 2C                         00010000
* CARD 600 OF FILE 2                       00011000
          ALLOW 2D,2A                      00012000
* CARD 700 OF FILE 2                       00013000
          ALLOW 2B,2C                      00014000
* CARD 800 OF FILE 2                       00015000
* CARD 900 OF FILE 2                       00016000
          END   XFILE2                     00017000
```

Control Card Input:

```
1          10          16                 73
$ID        NAME
$BAL                   LIST, ANALYZ
$SPT       MATCH       2A, 2C
$SPT                   SPT002, FILE2
           ALLOW       2E                 00003500
           ALLOW       2D                 00006000
           ALLOW       2A                 00008000
```

Assembler Output:



```
ADDR1  ADDR2   LINE SYMBOL     OP   OPERAND-COMMENTS      IDENT

000000         00001 XFILE2    START 0                   00002000
               00002 * CARD 100 OF FILE 2                00003000
               00003 * CARD 500 OF FILE 2                00009000
               00004 * CARD 600 OF FILE 2                00011000
               00005 * CARD 700 OF FILE 2                00013000
               00006 * CARD 800 OF FILE 2                00015000
               00007 * CARD 900 OF FILE 2                00016000
000000         00008           END   XFILE2              00017000
```

In example 2 the ALLOW cards were manipulated via the input control card stream.

Note 1: In both examples the special $SPT match card precedes the normal $SPT card. If the order is reversed, the BATCH function will be turned off upon detecting the dollar sign on the special match card.

Note 2: MATCH or MATCX once invoked remains in effect for the entire job. Thus, if regular "no MATCH or no MATCX" steps and MATCH or MATCX job steps are mixed in a job, the "no MATCH or no MATCX" steps *must* come first.

2.11.2 Tape to Print/Punch

The tape to print/punch capability of BATCH allows for either printing, punching, or printing and punching a utility monitor-generated SYSOUT. Other tapes can be printed in special formats as noted in the possible suffixes to the initiation message.

To initiate tape-to-print/punch operation, mount a tape to be processed on an available drive (7- or 9-track as appropriate excluding SYSOUT) and enter one or more of the following messages as appropriate, starting with a TTP message to select the processing mode and associate a tape drive with the print/punch group.

TTPp nnnaa (,dddd,mmmm)

Start to process the tape on drive nnn using print/punch group p. Options assumed are to start from present position and to print and punch to end of file. Mode of processing is controlled by field aa as follows:

(field missing or other than /S, /N, /X, /A, or /O) tape is assumed to be utility system SYSOUT format and will be processed according to carriage control fields in the tape records.

/S – tape is assumed to be utility system SYSIN format and will be printed single-spaced.

/N – tape is assumed to contain unblocked print images, which will be printed single-spaced.

/X – tape is assumed to be unformatted and will be printed in hexadecimal (hex/dump).

/A – tape is assumed to be unformatted and will be printed in alphanumeric characters (alpha/dump).

/O – tape is assumed to be an OS format SYSOUT tape and will be processed according to carriage control fields in the tape records if present. If not, records will be printed single-spaced. The maximum blocksize supported is 1160 bytes.

The part of the message (,dddd,mmmm) is density and mode for 7-track tapes other than SYSOUT. SYSOUT is assumed as density 556, odd parity, no byte converter. If these fields are not present, density 800, odd parity is assumed. The possible values for dddd can be D200, D556, D800. The values for mmmm can be:

BIN – The tape was written using a byte converter (utility system tape, Library, compool, SPT, or SYSIN)

EVEN – The tape was written with even parity.

EVENT – The tape was written with even parity in Binary Coded Decimal (BCD).

ODD – The tape was written with odd parity.

ODDT – the tape was written with odd parity in Binary Coded Decimal (BCD).

Some of the options associated with tape to print/punch are:

a. **ALLp**

Continue processing to end of tape.

b. **STPp**

Stop processing with print/punch group p. When restarting the print/punch, start at least two records in front of stopping point.

c. **NPRp NPUp PRTp PUNp**

Only the most recent of these messages is applicable to group p. NPR means no print but punch. NPU means no punch but print. PRT and PUN each mean both print and punch.

d. **BSRp ccccc**

Backspace record (about five lines per record). Character p is the printer/punch group with which the tape to be positioned is associated. Character ccccc, a decimal number of one or more digits (separated from p by one space), is the number of times the operation is to be executed.

e. **BSFp ccccc**

Backspace file – see BSR.

f. **FSFp ccccc**

Forward space file. – See BSR.

g. **FSRp ccccc**

Forward space record. – See BSR.

h. **RWDp**

Rewind.

i. **RUNp**

Rewind and unload.

j. **TRNp n**

Translate the tape being processed according to the requested EBCDIC character translation. Character p is the print/punch group with which the tape to have special translation is associated. Character n, a decimal number from 0 to 2 (separated from p by one space), is the translation table wanted. The possible values are:

(1) 0 – printed as if online (includes sense and retries)

(2) 1 – uses 48-character set translation table.

(3) 2 – Use the complete character set as found on the PL 1 print chain. Includes both (%, (; #, =; +, &; etc.) Note: This is assumed if TRN is not typed in.

BATCH types each $ID card processed and types EOF for each end of file and end of tape when completing an ALL request.

Note: The console typewriter should be monitored for any of the following messages. The message implies incorrect tape format (unlikely with utility system SYSOUT tapes), wrong tape mounted, possible hardware error (if associated with SENSE-RETRY messages), or the operation needs access to another SE.

BATCH prints the following messages on the printer involved as appropriate:

```
***********************BAD PRINT/PUNCH TAPE FORMAT****************
***********************OVERLENGTH PRINT RECORD********************
***********************OVERLENGTH TAPE RECORD*********************
```

Tape format does not conform to specifications for utility system SYSOUT or SYSIN tapes as appropriate, or the tape was processed with the second SE unavailable and record size exceeds buffer size.

The most common causes are:

a. Incorrect tape mounted; mount a SYSOUT (or SYSIN) tape.

b. SYSOUT tape was not rewound before being written by utility system, space forward one or more files until valid data appear.

c. Operator generating SYSOUT did not allow utility system to cycle through end of batch before unloading the SYSOUT tape. Consider tape completely printed.

Other possible causes are user error such as an incorrect call to utility system print routines or incorrect format on a tape not written by utility system but supposedly conforming to utility system rules.

2.11.3 Tape Log

Tape logging can be used to determine the number and size of records and files on a tape. The input message for tape log is in the format:

TLGp XXX(,dddd,mmmm)

where:

p – the number of the printer the output is to go onto.

XXX – the physical address of the tape.

dddd – optional density for 7-track tapes only (not required if density is 800 unless mmmm field is used). Possible values are:

D200
D556
D800

mmmm – optional mode for 7-track tape only (not required if ODD parity is desired). Possible values are:

BIN – the tape was written using a byte converter. (Examples are utility system, Library, compool, SPT, or SYSIN tapes).

EVEN – the tape was written with even parity.

EVENT – the tape was written with even parity in the Binary Coded Decimal (BCD).

ODD – the tape was written with odd parity.

ODDT – the tape was written with odd parity in Binary Coded Decimal (BCD).

For a 7-track tape with no density, or mode specified density 800 bpi; ODD parity is assumed.

The printer output is in the form:

FILE XXXXX; RECORD NUMBER XXXXX;
NUMBER OF BYTES (16) XXXXXX;
NUMBER OF BYTES (10) XXXXX;
NUMBER OF WORDS (10) XXXXX

For the first record of each file a hexadecimal and an alphabetic dump of up to 256 bytes is taken.

All of the options for tape to print/punch except (d) are valid for tape log and have the same meaning they do for tape to print/punch.

2.11.4 Tape Punching

Several special types of tape may be wanted on cards. This capability produces a punched deck from a SYSIN, .AUXIL, or any other tape. The message is in the format:

TPNp XXX(/c) (,dddd,mmmm)

where:

p – the number of the print group for the punch onto which the output is to go.

XXX – the physical address of the tape.

/c – for a special tape. This can have the values:

/T – punching an .AUXIL tape. The tape will be punched on loader format text cards.

/S – punching a SYSIN. The cards will be punched out as read in. They will also be listed unless NPRp is specified.

/N – punching an unblocked tape. The cards will be punched out as read in.

BLANK – The tape will be punched on loader format cards with one deck per tape file.

dddd,mmmm – refer to tape logging.

An .AUXIL tape is only one file; so, the end of file is assumed as the end of tape.

All of the tape to print/punch options are valid for tape punching except (d).

NPRp is valid for punching a SYSIN and it means to punch, but not print.

2.11.5 Tape Duplication

BATCH can duplicate any tape written according to the following rules:

a. Records may not be less than 16 bytes or more than 65,535 bytes long. (All records written under control of the utility monitor conform to this rule.)

b. A tape may contain any number of files but must have two consecutive tapemarks recorded to indicate end-of-tape. (Utility system, Library, compool, SPT, SYSIN, and SYSOUT tapes follow this rule.)

To duplicate a tape, mount it on any drive (except SYSOUT, which cannot be read). Mount a scratch tape on another drive (except SYSOUT, which cannot be read for checking). Be sure at least two SE's are available. Tape reading and writing will be unbuffered if two SE's are available, and buffered if three or more SE's are available. Type the following message as instructions to BATCH:

DUPxxx,yyy
or
DUPxxx,yyy,dddd
or
DUPxxx,yyy,dddd,mmmm

where:

xxx is physical address of tape to be copied. yyy is physical address of the tape or tapes to be written on. Where more than one tape is to be generated as the form yy1/yy2/yy3/etc. (For most efficient operation select a different channel for each consecutive unit specified.) dddd is density for any 7-track tapes involved (not required if density is 800 unless mmmm field is used). Possible values are:

D200
D556
D800

mmm is mode of reading or writing on any 7-track tapes involved. Possible values are:

field omitted – suitable for almost all 7-track to 7-track duplication.

(Exceptions are even parity tapes and tapes with records longer than 65,535 six-bit bytes. A typeout appears if this problem exists.)

BIN

required for duplicating between 9-track and 7-track when byte converter use is required. (Examples are utility system, Library, compool, SPT, or SYSIN tapes.)

Note: Utility SYSOUT tapes should never be duplicated to a different track format.

EVEN

required if 7-track tapes involved are even parity.

EVENT

required if 7-track tapes involved are Binary Coded Decimal (BCD) with even parity.

ODD

optional if 7-track tapes are odd parity. ODD is assumed unless designated otherwise.

ODDT

required if 7-track tapes involved are Binary Coded Decimal (BCD) with odd parity.

BATCH duplicates records and tapemarks to and including the double tapemark at the end of tape, rewinds and unloads unit xxx, and types one of the following messages:

```
FILE COUNT fff, RECORD COUNTxxxxx
FILE COUNT fff, RECORD COUNT xxxxx,
***WARNING*** yy
     RECORDS MORE THAN 65535 BYTES LONG
```

where:

ff is number of files copied

xxxxx is number of records copied

yy if present, is a warning of possible failure. If copying from 7-track tape without specifying mode and tape was originally written using byte converter (BIN), repeat specifying BIN mode. Otherwise, assume duplication failed unless the tape is known to contain yy records each exactly 65,535 bytes long.

Then, BATCH proceeds in one of the following ways:

a. More tapes are to be recorded. Reassign the first yyy (written just previously) as xxx and repeat the duplication.

b. The last tape has been recorded. Reassign yyy as xxx for a read only pass to validate the last tape recorded.

c. The read only pass is complete. Process the next operator request.

A special case of tape duplication is conversion of a 7-track SYSOUT tape from a 9-track SYSOUT tape. A straight duplication from the 9-track to 7-track will not produce the desired results so the following message is used:

SYS xxx, yyy

where:

xxx is the physical unit number of 9-track SYSOUT tape to be converted.

yyy is the physical unit number of the 7-track tape the converted SYSOUT data is to be written on.

All of the normal duplication restrictions hold.

When the SYSOUT conversion is completed the message

END SYSOUT CONVERT

is typed and the tapes unloaded.

2.11.6 Merged Library/Compool (MLC) Creation

To reduce the number of tape drives required for JOVIAL compilation, Library and compool data may be merged onto a single (MLC) tape.

BATCH can merge library data from a library or Merged Library Compool (MLC) tape with compool data from a compool or MLC tape to create a MLC tape. MLC tapes can then be mounted on LIB1 for use by the JOVIAL compiler or BAL assembler when there are insufficient tape drives to permit assignment of a separate .COMP drive.

Proceed exactly as for DUP except for control message format which is:

MLCxx1,xxc,yyy

where:

xx1 is the physical address of the tape with library data.

xxc is the physical address of the tape with compool data.

yyy is the physical address of the tape or tapes to be written in the same manner as for DUP.

2.11.7 Tape Comparison

The tape compare capability of BATCH can be used to determine the differences or lack of differences between two tapes. There are several limitations, restrictions, or special conditions for tape compare. They are:

1. There must be at least 2 SE's. For hex compares, tape reading will be unbuffered when two SE's are available, and buffered when three or more SE's are available.
2. For a compare of SYSOUT tapes, the tapes must be either both 9-track tapes or both 7-track tapes.
3. The compare is terminated by the end of one tape or by accumulating a number of errors which exceeds: 50 + (0.1 x number of records processed).
4. The file will be read out and compare suspended for 20 errors in a file.
5. When one tape reaches end of file the other will be read and printed out until it also reaches end of file.
6. For end of tape on one tape, the other tape will be read to end of file and then the compare is finished.
7. For SYSOUT tapes, the first header on each page and all punched cards are ignored for the compare.
8. For SPT tapes, the header record is ignored and comparison starts with file 2.
9. All EBCDIC tape compares will attempt to find a match for records which do not match. Both tapes are searched 25 logical records down the tape before a compare error is declared.
10. For comparison of SYSIN or SPT tapes, the last eight card columns (73–80) are ignored.

The input message for the first tape is:

CMP$_p$ XXX(/c) (,dddd,mmmm)

where:

p – the number of the printer which generates output.

XXX – the physical address of the tape.

/c – for a special tape. The possible values are:

/p – two SPT tapes to be compared
/y – two SYSOUT tapes to be compared
/s – two SYSIN tapes to be compared
BLANK – any two tapes to be compared.

dddd,mmmm – refer to tape logging.

The message for the second tape is:

CMPX XXX(,dddd,mmmm)

where:

XXX – the physical address of the tape

dddd,mmmm – refer to tape logging.

It is possible that comparison may be wanted starting with other than the first file.

To position a tape to be compared at other than the load point the message is:

POS$_p$ fff,rrrr

where:

p – the printer as assigned by CMP$_p$ or an X if CMPX is to be spaced.

fff – the file at which the compare is to start (zero, blank or 1 if the first file)

rrrr – the record within the file at which the compare is to start (zero, blank, or 1 if the first record).

The output will be found on both the printer and the typewriter. For two tapes which compare exactly, the message

****NO DISCREPANCIES**

is typed out.

The possible typewriter messages before the compare starts are:

(a) ****NOT ENOUGH INPUT FOR COMPARE ENTER REQUEST AGAIN –**

Both units for the compare were not defined. Just type the one needed and depress the ENTER key.

(b) **CMP$_p$ ILLEGAL POSITION REQUEST –**

An end of file was encountered while trying to position to the requested record. Type the correct position request and depress the ENTER key.

(c) *****CMP$_p$ BAD TAPE FORMAT. TYPE SKIP TO END COMPARE –**

The tape mounted on the specified unit is not in the format the compare message indicates. If the wrong tape is mounted, mount the· correct tape and enter. Type in "SKIP" to end the compare at this point.

The printer messages are:

(a) *****FILE 123; RECORD 12345; TAPE CMP$_p$ –**

This is the position of the specified tape when the error was detected.

(b) **LENGTH CMPn = yyyyy CMPX = yyyyy**

This message follows (a) when in a hexadecimal tape compare, the non-compare was caused by record length. n represents the print group the output is going to and yyyyy represents the place taken by the physical lengths of the two records.

(c) *****MATCH FOUND THROUGH TAPE POSITIONING –**

This message follows (a) when a match has been found for an EBCDIC compare. The record will follow this. This will then be followed by a printout of all records up to the one which matched.

(d) ****SKIPPED TO NEXT FILE, 20 ERRORS THIS FILE –**

The 20th error in a file has just been printed out. The tapes are spaced to the next file and processing continues.

(e) ****THERE WERE 10 MISMATCHED RECORDS**

This is the summary message which is printed and typed at the end of the compare.

In addition to the above messages up to 256 bytes in a hexadecimal tape compare or the logical record in an EBCDIC compare are printed.

The "ALLp" and "TRNp n" messages have the same meaning for tape compare that they do for tape to print/punch.

When the compare is completed by either end of file, if "ALLp" was not typed, or end of tape, if "ALLp" was typed, the tapes that were compared are unloaded.

2.11.8 Write End-of-File Tapemarks on Tape

BATCH writes end-of-file tapemarks on any tape. To accomplish this, mount the tape on any drive except SYSOUT. Type the following message as instructions to BATCH:

WTMxxx/n

where:

xxx is the physical address of the tape to be written.

/n is optional and, when included, designated the number of tapemarks to be written. One tapemark is written when /n is not included; n tapemarks are written when /n is included.

3.0 UTILITY SYSTEM TAPE GENERATION

3:1 INTRODUCTION

This section is intended to provide optimized procedures for utility system tape generation. To ensure all conditions are described, this set of procedures assumes new system tape generation rather than old system tape update.

New system tapes are produced to add new system processors or to change existing processors. Source statement changes or new source language subprograms generally require production of new SPT's and subsequent program compilation and/or assembly. The work flow for utility system tape generation includes the following steps:

a. Build new SPT's (Symbolic Program Tapes).

b. Use SPT's to compile and/or assemble off-line

c. Obtain absolute decks from SYSOUT tapes

d. Set up decks for system tape edit.

e. Run SYSEDT

f. Test the new utility system tape.

3.2 SPT PRODUCTION

After completion of subprogram debug and test, the source decks are cataloged on SPT's using the SPT edit subprogram (SPTEDT). Each subprogram constitutes one file on SPT; therefore, multifile SPT's are recommended to reduce the tape volume required. The arrangement of subprograms on the multifile SPT is optional; structuring the SPT so that its Subprogram sequence conforms to the job sequence saves time because it minimizes tape movement during language translation.

The SPT contains a tape label file and should contain a tape summary file, subprogram source statement files, autochart source statement files, and compool source statement files. The tape label content is optional; however, it is recommended that the creation date be part of the labeling convention to simplify update procedures. The contents of the tape summary file are also optional but should include such information as name, length, and other descriptive file information. Following the tape label and tape summary files are the subprogram source statement files. Autochart source statement files and/or compool source statement files can either follow the appropriate subprogram file or can be grouped following the subprograms.

3.3 LANGUAGE TRANSLATION

The symbolic program tapes produced from source statement decks and appropriate control cards are the input for compilation and/or assembly. The object deck output of language translation is either in executable (core image) form or in relocatable form. Those object decks in core image form can be cataloged directly on the utility system tape using the system tape edit subprogram; object decks in relocatable form require additional processing.

Object decks are in relocatable form when the subprogram contains external references (EXTRN's) or when the program is to be loaded at an address other than the address where it was assembled. It will be assumed in the document that all system processors are assembled at the address where they are to be loaded.

EXTRN's are generated for JOVIAL subprograms by the JOVIAL compiler and for BAL subprograms by the programmer. An EXTRN is required whenever the subprogram calls a separately assembled subprogram, calls a Library Routine, uses a compool, or references a symbol which is not defined in the same assembly.

Addresses for external references are supplied at load time by the utility system loader. Therefore, subprograms containing external references must be processed by the utility loader to obtain core image object decks (absolute object decks). The general sequence of events associated with absolute object deck production is:

a. Language translation off-line to produce a .SYSOUT tape and an .AUXIL tape containing relocatable object deck images.

b. Execution of the loader subprogram using the PUN card option to obtain executable object deck images on the .SYSOUT tape.

c. Punching executable object decks from .SYSOUT using the BATCH feature of the utility system monitor.

The following paragraphs supply a detailed description of input and output for language translation. Each description is associated with an appropriate language translation situation.

3.3.1 Jobs Without EXTRN's

The control card sequence for each job of this type is:

```
$ID
$NONEX
$BAL        LIST, PUNCH, ANALYZ
or
$JOV
$SPT        tape label, file name
7
8EOF
```

The output for each job is a subprogram listing and an absolute object deck ready for editing to the utility system tape.

3.3.2 Jobs With EXTRN's – No Compool Referenced

The suggested control card sequence for each job of this type is:

```
    $ID
    $XEQ        (This card causes the program to be
                 loaded after compilation and/or
                 assembly)
    $OBJ        (This card must precede the PUN
    12           card.)
     2PUN       (This card causes an absolute deck
     9           to be punched.)

    $BAL        LIST, PUNCH, ANALYZ
or
    $JOV
    $SPT        tape label, file name
    7
    8 EOF
```

The output for each job is a listing, an object deck, and an absolute deck. Selection of the PUNCH option on either the $BAL card or the $JOV card causes a relocatable object deck image to be produced in addition to the absolute object deck image. The relocatable deck is not necessary, but useful in case further debugging is required.

If the PUNCH option and the PUN card are both used, two object deck images are produced. The user must separate the relocatable deck from the absolute deck, retain the relocatable deck for future use, and continue processing using the absolute deck. If the PUNCH option is not selected, only an absolute deck is produced.

Note: Although the lowest address at which a subprogram can be loaded by the system loader is X‘COOO’, system processors can be loaded as low as X‘9000’. When using the system loader to obtain an absolute deck the user must specify the START card address in the PUN card. If the system loader cannot honor the start address it will physically place the subprogram at the first available address and output from that address, but the text card images are assigned addresses starting at the address specified in the PUN card.

3.3.3 Jobs With EXTRN's – Compool Reference

Jobs containing references to compool require special consideration depending on how the compool is used. The following cases describe the uses of the compool.

Case 1: Compool origin is at a lower address than the subprogram but does not contain preset constants.

Case 2: Compool origin is at an address higher than the subprogram.

Case 3: Compool origin is at a lower address than the subprogram and contains preset constants.

In cases 1 and 2 the absolute deck images are produced from the starting load address of the subprogram. The PUN card in case 1 should specify the same address as the start of the lowest addressed compool object deck. The PUN card in case 2 should specify the same address used in the subprogram START card. In case 3 the absolute deck images should be produced from the starting load address of the compool so that the preset constants will be included in the absolute deck. Therefore, the PUN card address specification must be the starting load address of the compool.

The suggested control card sequence for jobs with compool references is:

```
$ID
$XEQ
$OBJ
12
 2
 9 PUN
$JOV   LIST, PUNCH, ANALYZ, PUNCHC, LISTP

$SPT   tape label, file name
7
8EOF
```

The output for this type job is a program listing, a relocatable object deck, an absolute object deck, a punched deck image of compool segments, and a compool listing. The PUNCHC option on the $JOV card causes the compool segments to be loaded with the subprogram for the PUN run of the system loader. It also causes punched deck images of the compool segments to be written on .SYSOUT. Without this option, compool segment object code will not be placed on the .AUXIL tape and the program will encounter loading errors. The LISTP option on the $JOV card causes a BAL listing of the compool segments to be generated.

3.4 SYSOUT PROCESSING

The absolute object deck images on the .SYSOUT tapes must be converted to punched card form for the system tape edit subprogram. The BATCH function of the utility system monitor is used to make the conversion. The computer operator is instructed as to which .SYSOUT tapes are to be punched by use of a job request slip. After punching, the decks are separated by the subprogram using the names supplied in the $ID cards.

3.5 SYSEDT DECK HANDLING

Special handling may be required by certain absolute object decks before they can be used as input to the utility system tape edit subprogram. Absolute object decks directly produced by the assembler contain a $OBJ card which must be removed prior to the tape edit operation. Absolute object decks for single core loads that exceed X'FFFE' bytes in length are automatically split into X'FFFE' byte records by SYSEDT. The entire deck size must appear on the control card, for the parent component and SYSEDT generates record names for the overflow records. Absolute decks containing a compool consisting of preset constants origined at an address lower than the origin of the subprogram or absolute decks containing overlays can be sequenced for SYSEDT to facilitate transfer of control when the system processor is loaded for execution.

An understanding of utility system tape structure and of how the utility system monitor reads records from the system tape provides a basis for the deck handling requirements specified in the following paragraphs. The utility system tape consists of multiple files, each file consisting of system tape records functionally grouped. The utility monitor's absolute loader section reads the system tape, loads the processor into core storage, and transfers control to the first byte of the first record read.

When a subprogram contains a compool consisting of preset constants origined at an address lower than the subprogram, an adjustment must be made to prevent transfer of control to the compool at load time. Two methods are available to the user to ensure correct transfer of control. A programmed branch (to branch to the address of the first program instruction) can be patched into the deck at SYSEDT time using a REP card or the following procedure can be implemented:

a. Split the absolute deck separating the compool and the subprogram.

b. Place a completed INSERT card with an A in the continuation field in front of the subprogram deck to indicate a continuation record follows.

c. Place a completed INSERT card with an R in the continuation field in front of the compool deck and place the compool deck behind the subprogram deck.

d. Run SYSEDT.

The absolute loader section of the utility monitor loads the subprogram deck as the first system record even though it is origined at an address higher than the compool. The absolute loader then loads the compool; control is transferred to the first byte of the subprogram.

When more than one core load of a subprogram is necessary, each overlay is contained in a separate system tape record. All overlays of a particular subprogram must be grouped in a single tape file.

When a compool containing preset constants origined at an address lower than that of the first overlay is used, the first overlay is designated the first system record of the file and the compool is designated the continuation record (designation is done by use of INSERT cards). The remaining overlays are each designated as separate system records.

If more than one assembly is used in any overlay, all the absolute decks resulting from language translation can be put on a single tape record if the total length does not exceed X'10000' bytes. Continuation records are generated by SYSEDT for components which exceed X'FFFE' bytes.

3.6 SYSTEM TAPE EDITING

When all decks are ready, the remaining control cards for the system tape edit subprogram are added, and the SYSEDT job is run. The decks should be ordered so that the most fiequently used system processors are placed first on the utility system tape.

The system tape label should be meaningful to the user. Inclusion of the creation date is recommended as part of the user's labeling convention.

3.7 SYSTEM TAPE TESTING

After the system tape has been produced it should be tested. The user may select test cases for each system processor, run a series of jobs to exercise each processor, and inspect the output.

Appendix A. PROGRAMMER AIDS

A.1 INTRODUCTION

The information contained in this appendix is a collection of specific aids which should be of help to the programmer. If more specific information is desired, consult the appropriate User's Manual.

A.2 TRACING A PROGRAM THROUGH COMPILATION, ASSEMBLY, EXECUTION

A.2.1 Writing a JOVIAL Program

When a programmer intends to write a JOVIAL program, he should first design the program completely. The program is broken into logical tasks and then a detailed flowchart is drawn. Only after these have been done should the coding begin.

The programmer should be careful when coding his program to include meaningful comments and remarks. Remarks are made before each task or logical section in the program. These explain what the section is to do. Comments appear beside each instruction; they give an idea of what the instruction does. The programmer should take care that his listing is reasonably formatted because comments cannot extend beyond Column 66.

When the programmer feels that his program is of a reasonable size and is properly coded and formatted, he creates an SPT tape with his source program. This will eliminate the possibility of lost cards or shuffled decks.

A.2.2 Compiling

When the programmer has coded a significant amount of his program, he may begin compiling. The entire program need not be coded before it is compiled. For example, if only some data declarations were coded and keypunched, they could be compiled to uncover any errors in table or item format.

To compile a job, the programmer needs the following cards:

$ID
$NONEX or $XEQ

$$\left\{\begin{array}{l}\textbf{\$JOV}\\ \textbf{Source Deck}\\ \textbf{7/8EOF}\end{array}\right\}$$

The utility monitor reads the $ID, $XEQ or $NONEX, and $JOV cards. If the program is to be loaded and executed after compilation, the $XEQ card is used, and the monitor sets a switch to indicate that the program is to be executed. If the program is only to be compiled, then the $NONEX card is used.

The $JOV card informs the monitor that a JOVIAL source program follows. The monitor checks the options field on the $JOV card and sets the necessary switches. Then the monitor calls the JOVIAL compiler from the utility system tape. The compiler reads the source program deck and tries to convert it into assembler language (BAL).

There are four types of errors which the compiler may encounter.

1. Fatal – A fatal error (such as a system error) forces the compilation to be abandoned.

2. Major – An error of major severity causes the compilation to be abandoned after the phase of the compiler where the major error was encountered.

3. Serious – A serious error allows all compilation to be completed, but the program will not be assembled by the assembler.

4. Warning errors – These errors cause warning messages to be issued, but compilation continues.

The compiler processes and prints the input deck. If no errors of at least serious severity are encountered, the compiler puts its assembly language output on the .WORK1 tape to be read in by the BAL assembler.

The compiler calls the BAL assembler from the system tape. The assembler reads the assembly language input from WORK1. It generates object code from the BAL input. This object code is placed on the .AUXIL tape if the program is to be executed.

If the programmer supplied the LIST option on the $JOV control card, the assembler prints a listing of the

BAL input. If the ANALYZ option was selected, the assembler prints a listing which shows all symbols and references to them. If the PUNCH option has been selected on the $JOV card, the assembler would punch an object deck.

The assembler recognizes two types of errors: possible and serious.

1. Possible errors do not prevent the program from being loaded and executed.
2. Serious errors prevent execution of the program.

Let us assume that a program has been successfully compiled and assembled, and that the object code was placed on the .AUXIL tape by the assembler. The monitor would then call the utility loader from the system tape into storage. The loader would examine the object code on the .AUXIL tape and, if possible, would load the program into storage. Control would then be passed to the program. When the program completes execution (this is done via the STOP statement, the SYSCOM library routine, or a program interrupt), all debug requests (dumps and traces) are honored. Finally, the monitor regains control and continues to the next job.

A.3 HOW TO GET AN ABSOLUTE DECK

When a program object deck which contains EXTRN's is punched for a BAL or JOVIAL program, it is in relocatable format. This is not true for programs with no EXTRN's unless the programs are assembled at an address other than the address where they are loaded. The section assumes all programs are assembled and loaded at the same address.

To place a program on the system tape the program must be in absolute format: the punched deck is a core image of the program. Before the method of obtaining an absolute deck can be explained, it must be known that programs which require an absolute deck are those which contain EXTRN's. To obtain an absolute deck, the program must first be loaded so that all EXTRN's will be resolved. Then the core image of the program is punched. To do this, the following control cards are used:

$ID

$XEQ **(even though the program is not to be executed, the $XEQ card is used to indicate that the program must be loaded.)**

$OBJ **(This card must precede the PUN card.)**

12
2
9PUN **(This card causes the absolute deck to be punched.)**

$BAL
$JOV

source deck

7
8EOF

The format of the PUN card is as follows:

column	**1–4**	**7–12**	**14–17**
	12 **2** **9PUN**	**start address**	**processor name**

The start address is the address at which to begin punching. For example, if the program had a start address of X'9000', it would actually be loaded at X'C000'. If the PUN card contained an address of X'9000', punching would begin at X'C000' but relocation would be adjusted so that the absolute deck began at X'9000'.

The processor name is punched in columns 73–76 of the absolute deck,

A.4 HOW LIBRARY ROUTINES ARE REFERENCED

It may be to the programmer's advantage to understand the logic involved when he issues a call to a Library routine. For this example, it will be assumed that the program involved is coded in JOVIAL.

The programmer codes a call to a Library Routine in his program when his program is compiled; the JOVIAL compiler checks to see that the call references a real Library routine. The compiler then generates a BAL calling sequence to the Library routine. It also generates an EXTRN for the Library routine. The EXTRN is generated because the Library routine itself is not present until load time.

The program is then assembled and placed on the .AUXIL tape. At load time the utility loader loads the programs on the .AUXIL tape and also loads the Library routines from the Library tape. All EXTRN's are resolved and control is passed to the program.

Compilation – The Library tape must be present for compilation. The JOVIAL compiler reads the Procedure Descriptor Table from the Library tape so that it may validate Library routine calls, generate the calling sequences and EXTRN's.

Assembly – It is not necessary for the Library tape to be present during an Assembly since the BAL Assembler never uses the tape.

Load Time – If any Library routines are used, the Library tape must be present at load time. The utility loader uses the Library ESD Table to resolve EXTRN's and loads Library routines from the Library tape.

A.5 HOW EXTRN's ARE USED AND RESOLVED

It is often necessary for a program to reference a label which is not within the program itself. For example, if a call to a Library routine is generated in a JOVIAL program, the Library routine name is not defined within the program.

Any name referenced in a program which is not defined in the program must be identified by an EXTRN (external symbol). EXTRN's indicate to the BAL assembler that even though this symbol is not present at assembly time, it will be present at load time.

The JOVIAL compiler generates EXTRN's for the following conditions:

a. Calls to Library routines

b. Calls to separately assembled programs

c. Use of a compool (one EXTRN is generated for each compool segment referenced).

At load time there must be one entry point for each EXTRN. (This is not intended to imply that for each point there must be an EXTRN.) Entry points exist in the program where the symbol is defined. The one exception to this rule is a program name. If an EXTRN is defined in program A identifying the program name of program B, then program B need not contain an entry point for the program name. Program names are considered by the loader to be entry points.

An example of programs using EXTRN's and entry points follows:

```
Program A          Program B          Library

START PROGA        START PROG B       ZWPRINT

                                      ZVMVC

GOTO PROGB $

MVC (xx,yy,zz) $

TERM $
```

The JOVIAL compiler generates within Program A:

```
EXTRN      PROGB
EXTRN      ZVMVC
```

The BAL assembler cannot assemble object code for instructions that reference labels which are not defined in the program being assembled. Therefore, the assembler generates zeros in the object code where the address of the symbol would normally go. However, this address must be placed in the object code before the program is allowed to execute. Replacing this address is the job of the utility loader.

The utility loader is responsible for resolving EXTRN's. When the loader is called from the system tape by the utility monitor, all programs are on .AUXIL in object format. The loader first reads a table of Library routine names from the Library tape (if one is used). It then scans the programs on the .AUXIL tape and builds a table of entry points and EXTRN names and their addresses. The loader then resolves the EXTRN's by replacing the zero fields of instruction with the correct addresses. Now the program(s) can be loaded into storage for execution.

A.6 HOW THE COMPOOL IS USED

This section is most beneficial if discussed in two subsections: how to compile using a compool, and how to load and execute using a compool.

A.6.1 Compilation with a Compool

The programmer must be sure that the START card indicates that a compool is to be used.

The compool tape must be present on the .COMP tape drive or an MLC (Merged Library Compool) on the .LIB tape drive for compilation.

The options PUNCHC and LISTP may be used on the $JOV control card. The PUNCHC option will cause compool segment object decks to be punched and the LISTP option will cause a BAL listing of the compool to be generated.

A.6.2 Loading and Execution with a Compool

To execute a program using a compool, the programmer must be sure that both his program and the compool segment object decks are on the .AUXIL tape before being loaded.

There are three (3) ways to put compool segment object decks on .AUXIL. For these examples it will be assumed that the jobs are to be compiled, assembled, loaded, and executed.

1. If there is only one deck in the job (the deck to be compiled), the PUNCHC option must be selected on the $JOV card. This will cause the compool segment object decks to be placed on .AUXIL. Without the PUNCHC option only the program object code will be placed on .AUXIL, and loading errors will result.

2. If the programmer wishes, he can place the compool segment object decks (obtained from a previous PUNCHC run) in the job input stream. Then the PUNCHC option would not be included on the $JOV card.

3. Sometimes it is desirable to load only those compool segments which are used by the program so that a minimum amount of core is used for execution. To do this the programmer will include only those compool segment object decks which his program references. It is usually necessary to load in this manner since some compools are so large that all segments cannot be loaded in available storage.

   Of course, the program deck is included in the input stream. For this case, where some segment decks are omitted, it is necessary for the programmer to provide a dummy deck so that the EXTRN's for the omitted segments will be resolved. The programmer may do this by creating a small assembly deck with omitted segment names as entry points and labels.

A.7 SETTING END-OF-FILE AND ERROR RETURNS FOR TAPES

Whenever a programmer is to perform some I/O function using tapes, he must consider the end of file or error conditions which may occur. The programmer usually provides routines to handle these conditions; however, if desired, the utility monitor will assume control under these conditions.

There are two methods through which the programmer can provide for the routines to handle end of file of errors encountered on tapes:

1. By interrogating output parameters of Library routines.

2. By setting returns for tape units.

A.7.1 Method 1

Since the JOVIAL language does not provide input/output instructions, Library routines are used to specify I/O. Most Library routines provide output parameters which may be interrogated by a program to determine if an end of file or error condition was encountered.

For example, if the programmer wished to read from a tape unit, he may use the REED Library Routine and check for end of file or error. An example of the code he would produce is:

```
REED (10, TABL, 500=BDITM) $

IF BDITM EQ 1 $

GOTO EOFRT $

IF BDITM EQ 2 $

GOTO ERROU $

GOTO GOOD $
```

In this example the program reads from logical unit 10 into an internal area (TABL) not more than 500 bytes of data. If an end of file was encountered, a branch is taken to a routine labeled EOFRT. If an error was encountered, a branch is taken to the routine labeled ERROU. If neither an end of file nor error was encountered, a branch is made to routine labeled GOOD.

If the programmer intends to interrogate output parameters of Library routines in the manner shown above, he must issue a call to the SETRET Library routine prior to the initial use of any of those Library routines. The SETRET routine initializes tape units and allows the programmer to interrogate output parameters from Library routines using these tape units. Unless the SETRET routine is used, an end of file or error condition will cause control to pass the utility monitor and the job will be terminated.

A.7.2 Method 2

The programmer may wish to specify end of file and error return addresses prior to the initial call to an I/O Library routine. Then if an end of file or error condition is encountered on the specified unit, control is passed to the user's specified routine. This is done by using the SETRET Library routine. An example of this method is:

```
SETRET (6, EOFIT, ERRIT) $

SETRET (10, TIFOE, TIRRE) $

REED (6, INPT, 24=) $

RYTE (10, INPT, 24=) $
```

The items EOFIT, ERRIT, TIFOE, and TIRRE are defined in the program and each item contains an address. The coding shown above results in the following action:

a. Logical unit 6 and logical unit 10 have returns set for them.

b. The program reads from logical unit 6 into location INPT for a length of not more than 24 bytes.

c. If an end of file was encountered, control is passed to the address specified in item EOFIT; if an error was encountered, control is passed to the address specified in item ERRIT.

d. The RYTE routine causes 24 bytes of information from location INPT to be placed on logical unit 10.

e. If an end of file is encountered, control is passed to the address specified in item TIFOE; if an error is encountered control is passed to the address specified in item TIRRE.

f. The SETRET routine takes precedence over the checking of output parameters (i.e., if an end of file or error condition is encountered and the SETRET routine was used for that particular unit, control is passed to the appropriate routine before the interrogation of output parameters is executed.

g. SETRET may not be used for the system tape, SYSIN, or SYSOUT.

A.8 WRITING PROCEDURES AND FUNCTIONS IN A JOVIAL PROGRAM

It is often desirable for a JOVIAL programmer to include closed compound procedures, functions, and/or procedures within his program. These routines are usually coded to perform a particular service for the program. For example, some programmers prefer to do all printing within one routine instead of having many printing instructions interspersed throughout the program. Routines like this, with one purpose, are coded in JOVIAL in the form of closed-compound procedures, functions, or procedures. These are described generally below and in detail in the JOVIAL Language User's Manual.

A.8.1 Closed-Compound Procedures

This type of routine contains no input or output parameters. It performs a service and usually is called from several places in the program. A programmer would include a closed-compound procedure in his program when he wants a certain service provided, but there are no variables involved. For example, suppose every time a program was to type a message, certain items were to be initialized to zero. Then a closed-compound procedure would be practical to type the message and initialize these items.

A.8.2 Functions

A function is a routine which contains input parameters and produces a value. A function call is contained within a JOVIAL statement. For example, if REVRS were a function which provided the two's complement of an integer item, the function may be called by:

```
IF REVRS (ITM1) EQ NEG $

GOTO PSTV $
```

Likewise, the function may be called by:

```
NEGTV = REVRS (ITM1) $
```

In either case, the function is called and returns a value. The GETLIO Library routine is an example of a function.

A.8.3 Procedures

Procedures are routines which receive input parameters and may produce several output parameters. Most Library routines are procedures (e. g., REED, SETRET, RYTE). Procedures are used in a program when there may be several input parameters and a certain function is to be performed other than just producing an output value. For example, a procedure to print a line may be coded with input parameters to specify the address from which to print and the length of the print image. The output may consist of one parameter specifying whether or not the line was printed.

A.9 HOW THE COMPILER HANDLES INPUT/OUTPUT PARAMETERS

The first statement in a procedure or function is a PROC statement. This statement contains the names of the input-parameters, and in the case of procedures, the output parameters. The calls to procedures and functions also contain the names of input and output parameters. The parameters specified in the call are defined outside of the function or procedure while the parameters specified in the PROC statement are defined internal to the procedure or function.

When the compiler encounters a call to a procedure or function, it stores the input values in the input parameters of the PROC statement. Then it allows the procedure or function to operate. The procedure return is made and the compiler stores the output parameters from the PROC statement into the output parameters in the call, then processing continues with the next statement.

Appendix B. GLOSSARY

ASCII – American Standard Code for Information Interchange.

assembler – The system component that translates a BAL source program into loader language.

BAL – Basic Assembly Language.

batch – A collection of jobs that are intended to be executed serially in a single computer run; these jobs constitute the system input file.

block – A collection of contiguous data, covered by one base register.

blocking factor – The number of logical records that make up one physical record.

canonical form – An irreducible, ordered representation.

character position – The location of a character within a table; character positions are counted from left to right within the table, counting the first as zero.

common storage – A special section of storage that enables programs within a job to define a common storage area and manipulate data within it. The address of common storage is relocatable by the loader.

compiler – The system component that translates a JOVIAL source program into BAL.

compool – A collection of data declarations that can be used by a group of programs (communications pool).

compool edit program – The system component that creates and maintains the compool tape.

control card – A punched card containing information that indicates to a system component the nature of the job or operation to be performed.

correction cards – Punched cards used to merge, delete, replace, and insert statements in a source program recorded on the SPT; these cards may be used at assembly time to make temporary alterations, with the SPT edit program to make permanent alterations, or with the BATCH processor to create a SYSIN tape containing temporary alterations.

data processing system – The physical machine configuration of a computing system, including a central processing unit, storage, and input/output units.

debug edit – The system component used by the debugging system to print out traces and storage and/or tape dumps in the format requested by the programmer.

debugging system – The system component that handles execution-time debug requests submitted by the programmer, emergency dumps when a job cannot be successfully completed, and post-execution dumps.

down – Describes an input/output unit that is not available for use because it is inoperative, disconnected, etc.

EBCDIC – Extended Binary-Coded Decimal Interchange Code

end-of-file return – The address to which control is transferred when an end-of-file condition is detected on an I/O unit.

entry point – A symbol that is defined in one program and can be referred to by another program. Entry points are defined to the assembler by an ENTRY pseudo-operation.

error return – The address to which control is transferred when an error condition is encountered on an I/O unit.

execution time – The period of time during which any program is being executed.

external symbol – A symbol used in the operand of a statement in one program, but defined in another program. The symbol is an entry point in the program in which it is defined. External symbols are defined to the assembler by an EXTRN pseudo-operation.

file protection – A condition indicating a reel of tape that can be read only.

function – A defined procedure that produces one value each time it is called by the using programs. It is called from within a JOVIAL statement.

interrupt control – The portion of the monitor that handles any interrupts that may occur during processing.

interrupt stack – An area in the communications region where the PSW is stored when an interrupt occurs.

IOPACK – A collection of monitor routines that process input/output requests for problem programs and system components.

job – One or more programs which are intended to be executed as a single unit; each of the programs may require compilation or assembly prior to loading and execution.

job control – The portion of the monitor that performs system loading, control card logic, and input/output unit assignment.

library – A collection of IBM supplied routines that can be used by the system and the programmer; the programmer may write JOVIAL routines and BAL programs to be added to the library by using the library edit program.

library edit program – The system component that creates and maintains the library tape.

LISTIO – A parameter block of data and input/output unit information about units being used by a job; there must be one LISTIO for each unit.

loader – The system component that places object programs into storage for subsequent execution; in doing this, the loader assigns storage addresses, prints a map of storage used, loads any requested library routines, and furnishes linkage for inter-program references.

logical record – A logical unit of information contained within one physical record.

MONIO – A collection of monitor input/output machine-oriented routines that maintain asynchronous operations on all units receiving requests through IOPACK.

monitor – See utility monitor.

object deck – A punched card deck containing the loader language object code for a program.

object program – A program in loader language.

phase – A logical portion of a processor that contitutes one storage load.

physical record – All of the information recorded between two interrecord gaps on a tape.

position header – The location of the assembled address of an address constant that must be relocated.

problem program – A user program submitted to the 9020 Utility Programming System for processing and/or execution.

procedure – A series of JOVIAL statements that may be entered from the using program by a call, and, upon completion, return control to the point of departure.

process file – The basic unit of information that the compiler creates when translating the source program.

processing – Preparation of a program for execution, involving one or more of the system processors (see processor).

processor – The system components that provide services for the programmer; i.e., the JOVIAL compiler, the assembler, the utility leader.

production program – A program that has been debugged and operates according to the objectives established by the programmer.

real-time processing – Processing affected by external activities occurring at the same time as the execution of the program.

relocation header – The identity of a symbol whose relocated address must be supplied.

segment – As much of the process file as the compiler can handle for analysis at one time. Also, one of possibly several distinct portions of a compools.

source deck – A punched card deck containing BAL or JOVIAL source language statements to be processed by the assembler or the compiler and the assembler.

source program – A program written in either BAL or JOVIAL.

SPT edit program – The system component that creates and maintains the symbolic program tape.

symbolic program tape (SPT) – A tape used to store card image data. Programs stored on the SPT may be altered temporarily by BATCH to create an input tape, by the assembler during assembly, or permanently by the SPT edit program.

system edit program – The system component that creates and maintains the system tape.

system maintenance job – A job that involves creating or maintaining one of the tapes used by system components; i.e., the system tape, the library tape, the compool tape, and the SPT.

system tape – The tape that contains all the system components.

utility monitor – The system component that supervises the processing of all jobs, handles interrupts, and regulates input/output.

utility programming system – The programming support package for the IBM 9020 Data Processing System.