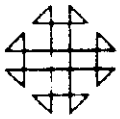
SHARE PROGRAM LIBRARY AGENCY

PROGRAM NUMBER

030014 ?

University of Miami

1365 MEMORIAL DRIVE - CORAL GABLES, FLORIDA
(305) - 284-6257



CONTRIBUTED PROGRAM LIBRARY SUBMITT
(for IBM S/360, 1130 and 1800)

SHARE Program Library Agency
Triangle Universities Computation Center
P. O. Box 12076
Research Triangle Park, N. C. 27709

This form should be completed and submitted with the program package to PID at the address shown above. Standards and instructions for submitting programs are in your *User Group Reference Manual* or the *Contributed Program Submittal Standards Manual* available from PID.

(1) Program Order Number *(to be filled in by PID)* 360D-03.0.014

(2) System Type *(machine)* S/360

(3) Search Key . . . . . 360 MOD 44 MULTIPROGRAM
MING SYSTEM / FOR / REAL TI
ME DATA ACQUISITION

(4) Programming Language . . . . . 360 44 PS

(5) Author's Name and Address . . . . .

Dr. W. M. Sachs
IBM Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

(6) Direct Inquiries to Name and Address *(if different than Author)*

(7) Title of Program . . . . Multiprogramming System (MPS)

(8) Submitter's User Group Affiliation Code and Installation Code . . . . . S YU

(9) Submitter's Own Program Identification and Suffix *(optional)* . . . . .

(10) Primary Subject Code 03.0

(11) Secondary Subject Codes 03.2 05.2 06.8

(12) Operating or Monitor System Required . . . . . NONE

(13) New or Revision Code *(if revision, show prior Program Order Number in item 1)* . . . . . N

(14) Year Completed 69

(15) Date of Submittal 10 69

(16) Documentation *(number of original pages submitted)* . . . . .

(17) **Abstract** *(should contain sufficient information for a reader to determine the value of the program). Listed on the reverse side of this form are subjects which may serve as a guide for a descriptive abstract.*

CONTRIBUTED PROGRAM LIBRARY SUBMITTAL FORM

Subject Guide

a. Purpose
b. Programming Language used
c. Version and modification level or release number of IBM Programming System used, or program order number for non-IBM authored program used
d. Field of application
e. Type of routine (main program, subroutine, etc.)
f. Specific description of machine requirements
g. Engineering Changes (EC) level of equipment (if pertinent)

ABSTRACT

MPS is a multiprogrammed operating system for the 360 model 44. The system, designed for real time data acquisition, supports multiple users in a fully protected environment. Features of the system include dynamic storage allocation, reentrant supervisor, Fortran compiler, loader editor job control processor, virtual device utilities, intertask communication facilities and operator control programs. The system requires at least 64K bytes of core, floating point feature, storage protection, reader, punch, printer, one SDSD, one other random access drive and one tape for system maintenance.

(Please attach additional pages if necessary) . . . . . **Total pages attached** _____

Permission to Publish

"I hereby give anyone permission to reprint, reproduce, and distribute this program to anyone else."

(18) **Signature of Submitter and Date** Robert Sachs Dec 9, 1969

(19) **Signature of Installation Addressee** Edward J O'Neill Dec 9, 1969

T4SF

Magnetic Tape Key

This volume contains one file formatted as follows:

800 bpi ; 9 track
Symbolic Card Images
EBCDIC
Sequence Numbers in cc 73-80; three letter prefix in cc 73-75
Irregular Blocking produced by 360 **44 RAP**
Maximum block size 2000 characters

The tape was produced by the 360 model 44 assembler's update facility. A listing of the modules prefixes may be obtained by assembling the comments module prefix ZZZ. A maintenance assembler is provided on the tape and should be used in the assembly of all other modules. The maintenance assembler (prefix ZCA and ZCB) should be assembled using 360 44 RAP and link edited using the instructions in comments module prefix ZAD. A sample deck using the maintenance assembler is contained in comments module prefix ZAB. A sample job may be punched out by assembling module prefix ZBA with the UPDATE1 option and accessing SYS003 to the punch.

DISCLAIMER

Triangle Universities Computation Center (TUCC) serves solely as the distribution agent for contributed programs and does not test or maintain them. They are distributed essentially in the original form submitted by the author. Neither TUCC nor SHARE, INC., makes any warranty, expressed or implied, as to the documentation, function, or performance of the contributed programs.

WRIGHT NUCLEAR STRUCTURE LABORATORY

YALE UNIVERSITY

INTERNAL REPORT NO. 38

User's Guide to MPS/44, a Multiprogramming System for the System/360, Model 44 Computer

Martin W. Sachs

October 1, 1969

PREFACE

This report is a general description of the features of the MPS/44 Programming System for the IBM 360/44 Computer from the viewpoint of the user of the system.

Since the MPS/44 system is an extension of 44PS, the standard programming system for the 360/44, a knowledge of the literature of 44PS is assumed, and, in general, information common to MPS/44 and 44PS is not repeated here.

The information herein, together with the 44PS literature, should be adequate for FORTRAN programmers. Assembly language programmers are frequently referred to the MPS/44 Design Notebook for additional details.

This report will be periodically updated by means of replacement pages on which the corrections and additions will be marked by means of vertical bars in the left margin. Replacements will be distributed to all active users at Yale and to the responsible authorities at other installations known to be using the system.

Some of the features of the system as described herein are not operational yet. These are listed in a postscript following the appendices. Improvements in this area will be announced by distributing revisions of the postscript.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

MPS/44 is a multi-user, multi-priority multiprogramming operating system for the IBM System/360, Model 44 Computer. It is designed to support multiple simultaneous users, each at an unique priority level.

The purpose of the system is to provide both extensive parallel processing facilities for individual applications, and to make maximum use of the computing power available by enabling more than one user to execute a program whenever a single user does not make full use of the available computing power. This is particularly important when one of the users is a real-time data acquisition program involving days of running at low input data rates.

Features of the system include dynamic storage allocation, a re-entrant supervisor, language and device independence, dynamic priorities, virtual device utilities, and extensive inter-user communication facilities. Both hardware and software mechanisms are used to protect users from each other. In addition to normal computational and data processing applications, the system is able to support, in a straightforward way, real-time, on-line data acquisition, apparatus control and monitoring, and both graphic and typewriter interaction between the user and his program.

Conceptually, MPS/44 may be considered to be a superposition of the multi-programming facilities upon the facilities found in the standard 360/44 programming system (44PS) such that to an individual user the system appears to be a standard 44PS batch processing system. As a result, familiarity with the literature of 44PS is assumed in this description, and the material contained herein is primarily a description of the facilities introduced by MPS/44.

The general concepts embodied in MPS/44 are described in the following literature:

a) M. W. Sachs, D. A. Bromley, M. Mikelsons, P. D. Summers, and J. Birnbaum, A Multiprogrammed Operating System for Nuclear Physics Data Acquisition, Presented at the IEEE Nuclear Science Symposium, Montreal, October 23-25, 1968. Published in IEEE Trans. Nucl. Sci. NS-16, 147 (1969) and as Yale University Wright Nuclear Structure Laboratory Internal Report No. 34.

b) M. Mikelsons and P. D. Summers, A Multiprogramming System for Real-Time Applications. Proceedings of the Skytop Conference on Computers in Experimental Nuclear Physics, March 3-6, 1969.

c) M. Mikelsons, A Flexible Task Scheduling Scheme for a Real-Time Environment, ibid.

d) Douglas T. Ross, The AED Free Storage Package, Communications of the A.C.M. Vol. 10, No. 8, August 1967, page 181.

e) MPS/44 Design Notebook.

The following literature on 44PS (with IBM form numbers) is relevant.

a) C28-6515-IBM System/360 FORTRAN IV Language.

b) C28-6813-IBM System/360 Model 44 Programming System - Guide to System Use for FORTRAN Programmers.

c) C28-6812-IBM System/360 Model 44 Programming System - Guide to System Use.

d) C28-6810-IBM System/360 Model 44 Programming System Concepts and Facilities.

e) C28-6596-IBM System/360 FORTRAN IV Library Subprograms.

f) C28-6811-IBM System/360 Model 44 Programming System Assembler Language;

g) C28-6814-IBM System/360 Model 44 Programming System - Systems Programmer's Guide.

MPS/44 was developed as a collaborative effort of the IBM-Thomas J. Watson Research Center, Yorktown Heights, New York, and the Yale University Nuclear Structure Laboratory, New Haven, Connecticut, with considerable assistance from the 360/44 group at the Space Radiation Effects Laboratory, Newport News, Virginia.

II. LOGICAL USERS

The basic unit of execution priority assignment in this system is the logical user.

A logical user consists of a job or sequence of jobs in the input stream bracketed by header and trailer cards described elsewhere. The jobs in the logical user are executed sequentially in the manner of 44PS but in parallel with other logical users' jobs.

Each logical user has a unique execution priority. In the present version of the system, priorities are determined on a first-come, first served basis. At any given instant, the highest priority logical user able to make use of the central processor has control over it. Whenever an interrupt (external, input-output, timer, program check, or some supervisor calls) occurs, the status of the various users is examined, and control goes to the highest priority user able to use the CPU (central processing unit).

Whenever the user in control at the CPU has to wait for an event (e.g. completion of an I/O (Input/Output) process) another lower priority user is given control until the higher priority user is ready to use the CPU.

The system components which determine user priorities and execution scheduling are modular, and hence easily replaced by different algorithms, should the need arise.

It is also possible to rearrange priorities during execution, by means of suitable operator commands.

During execution, a logical user may be in any one of the following states:

a) Dormant - waiting for availability of a system resource.

b) Waiting - execution has begun and the user is now waiting for completion of I/O activity or occurrence of some event.

c) Active - user is either currently in control of the CPU or has been interrupted by a higher priority user.

III. PHYSICAL USERS; STORAGE PROTECTION

The 360/44 includes memory protection hardware which enables any program or data area to be automatically protected against errors or unauthorized access by other programs.

In order to allow maximum protection while still enabling a single application to make use of the parallel processing capabilities of MPS/44, the definition and use of protection is handled separately from logical user management.

The basic unit of protection is known as a physical user. The physical user consists of one or more logical users plus optional data areas (one of which may be a global symbol zone) all of which share the same unique protection key. Within a physical user the logical users may communicate at full core storage speeds via the common global symbol zone. They may thus operate in parallel on the same data, if desired.

Since physical users have unique protection keys, they cannot directly access each other or each other's data. Any such attempt results in immediate job termination. Similarly the system resident supervisor and essential central information are totally protected against users' errors, to eliminate the possibility that the whole system will "crash" due to user errors.

It can be seen that physical users in general correspond to the real people using the system.

Although all the logical users in a given physical user may be entered into the system sequentially in the group, and hence occupy adjacent priority slots, in fact this is not necessary, and it is thus possible to organize the priorities of various operations without regard for their physical users.

Physical users are set up by means of commands entered through the console typewriter. This procedure is described in Chapter XII-C.

IV. DATA SET MANAGEMENT

The system supports all types of 44PS data sets, in their 44PS formats. In addition, a new type of data set, called a queue, is made available, as described elsewhere.

Each data set is dedicated to the user who accesses it until the end of the job in which it was accessed. During this period, no other user may access it. The only exceptions are the system phase and module libraries which are available to all through the loader editor.

Later versions of the system will include a generalization of the data management procedures which will enable data sets to be accessed by more than one user.

All standard data sets have a fixed expiration date, 99999. This enables disk volumes to be processed with the 44PS SQUEEZE utility.

A user who needs a data set other than a queue for only part of a job should release it as soon as possible by means of the //UNLOAD statement or an UNLOAD supervisor call within his program. Unless this is done, the data set does not become available to others until the end of the job.

Data sets on the system residence volume are intended only for use by system programs (such as the loader-editor) and may not be mentioned in any job control statements.

V. QUEUES AND VIRTUAL DEVICES

A. Queues

In order to simplify the allocation of work data sets only needed within a job or job step, MPS/44 makes available a new type of data set, called a queue. Space for the records in a queue is dynamically allocated as needed, in a special disk data set.

The queue is so called because it behaves in a manner analogous to a queue. Records are always written onto the end of the queue and read from the beginning. Once a record has been read, it is no longer available (as with a card).

Queues are single-user data sets. If different logical users define queues with the same name, they still have their own separate queues.

One of the most common uses of queues is as data sets for the intermediate work unit (SYS001) of the FORTRAN compiler and loader-editor, both of which use SYS001 in a manner appropriate to a queue.

A queue is created with an ALLOC card. The LABEL card is used to inform the system of the maximum record size expected. Actual record sizes may be mixed. Smaller records are packed on the disk, unlike with normal data sets. The largest possible record size is 720 bytes. Once allocated a queue exists until the logical user is deleted. Following the ALLOC, other units may be accessed to the same queue by using its data set name. LABEL should not be used with these accesses.

Some of the supervisor calls and corresponding FORTRAN statements have special meanings, as follows:

READ: If a READ is issued for an empty queue, the following CHECK will cause the program to wait until a record appears in the queue (thus simulating the action of a card reader). In FORTRAN, the CHECK is implied in the READ statement. If necessary, a NOTE should be issued before the READ to check for data. A count of 0 is valid and means skip over the next record (thus destroying it). The I/O error codes have their usual meanings. If the pseudo-end-of-file record is read, an end-of-extent code is returned, but reading can continue at

the option of the calling program.

NOTE: Returns the number of records currently in the queue (including end-of-file records).

POINT: ignored.

REWIND: ignored.

BACKSPACE (FORTRAN): ignored.

WEOF (and FORTRAN-END FILE): causes the core buffer for the queue to be written onto the disk even if not full, and writes a pseudo-end-of-file record in the queue which will produce an end-of-file condition when read. Writing can continue after the WEOF.

UNLOAD (including both SVC UNLOAD and the //UNLOAD job control statement): closes out the queue and makes it available to the virtual device utility program. Unlike with a normal data set, it does not logically disconnect the unit, but rather starts a new queue. This statement would normally be used only with a queue which is intended to be printed or punched (i.e. a virtual printer, which see below).

If any records remain in a queue when its user is deleted, these records will be printed.

To empty a queue, first do a NOTE to obtain the record count, and then execute that many READs with zero byte count.

To insert records into a queue (including a virtual reader) at any arbitrary location, "rotate" the queue to the desired place, write the records, and rotate the queue around to its original position. Among the uses of this technique is generation of job control cards for later processing. The rotation is carried out by successively reading records and writing them back, using the record count provided by SVC NOTE as a guide.

Examples of use of queues:

a) To create a queue named ELIYALE, simultaneously access SYS002 to it, and specify up to 120 byte records:

```
//SYS002 ALLOC ELIYALE,QUEUE=
//    LABEL 120
```

The label field (SYSxxx symbol) is optional, as on any ALLOC card.

b) To later access SYS003 to the same queue:

//SYS003 ACCESS ELIYALE

B. Virtual Devices

Since there is usually only one each of printer, card reader, and card punch in the system, and they must be available to all users, the concept of virtual devices is introduced into the system as a special case of the queue.

Programs are written as if they had direct access to the printer, card reader, and card punch, but in reality they are working with disk queues which are filled from or output to, the real I/O devices at appropriate times. A program called the virtual device utility program handles this data flow in a way which is basically fully automatic, but which has options available to the system operator.

On input, the entire logical user deck is copied onto the disk before execution begins.

On output, each logical user's output is queued on the disk for later printing or punching. The first logical user's printing and punching begins when the logical user is deleted. The order of output may be altered by the operator using procedures described in Chapter XII-D.

It must be noted that although the virtual devices are actually disk data sets, the only permissible operations on them are those which are permissible for the real devices. It is, for example, impossible to backspace any of these devices. The only exceptions to this rule are that it is possible to read the printer or punch, and write to the reader. The meaning of this is the same as for queues: Writing to the reader adds cards to the end at logical user deck; reading the printer or punch removes records from the beginning of the queue. These records will not be printed or punched.

The meanings of the various supervisor calls and FORTRAN I/O operations are the same as for queues. Note that UNLOAD may be executed for the standard system output units (SYSOPT, SYSPCH, and SYSLST) with their normal accesses (e.g. SYSOPT accessed to the printer) to force early printout without logically disconnecting them. Writing can continue after the UNLOAD without a new access card.

Printer carriage control and punch stacker control are controlled with

LABEL cards exactly as in 44PS. Thus for non-standard accesses, carriage control is not used unless a // LABEL ,CTLASA card is included. For standard accesses, carriage control is in force unless turned off by an explicit access card with no // LABEL card, e.g. //SYSOPT ACCESS SDSOPT. Only the first LABEL card for a given device applies. Subsequent ones are ignored. This means that the same carriage control is in force (or not) for all units accessed to this device.

If carriage control is in force, and a character which is invalid for carriage control is encountered in column 1 of a record, processing is switched to non-carriage control (single space) for the rest of the queue.

Each printer or punch queue is identified by a suitable header record.

On the 1442 reader-punch, punched output is handled as follows: Immediately following IPL, disable punched output. Whenever there is a punch queue to be output, disable reader input and enable punch output. A by-product of this procedure is that programs involving intermixed card reading and punching can be executed with a 1442 without using an intermediate data set since the virtual device mechanisms implicitly provide such a data set.

The normal "default" accesses for the system units SYSIPT, SYSRDR, SYSOPT, SYSLST, and SYSPCH are to the various virtual devices, which have symbolic names SDSIPT (standard input), SDSRDR (job control input), SDSOPT (standard output), SDSLST (system control output), and SDSPCH (punched output), respectively. They may be re-accessed to other types of data sets in the usual manner.

Other output units may be accessed to the virtual devices, either with their output intermixed with or separate from that of the standard units.

Additional printers, punches, and readers may be set up with ALLOC cards, as shown in the examples following. In this way the output from different symbolic units may be kept separated instead of intermixed, just as if there were several real printers in the system. If desired, LABEL cards may be used with these ALLOCs to shorten the standard line length (record size).

The use of an additional reader is this: The reader may be ALLOCed,

and filled with data by a program. After the program unloads it, it is treated as a new logical user deck and will be set-up and processed just as if it had been entered through the card reader. Such a reader should not contain the 12-7-8-9 header and trailer cards. This unit can be unloaded several times to create several logical users.

a) To access an additional unit to the printer, with it output intermixed with SYSOPT and SYSLST:

```
//SYS004 ACCESS SDSOPT
```

or

```
//SYS004 ACCESS DS,SAME=SYSOPT
```

b) To define several virtual printers such that their outputs are separate instead of intermixed, for example:

```
//SYS003 ALLOC P1,PRINTER=
//  LABEL  ,CTLASA
```

This causes the SYS003 output to be printed separately from SYSOPT, with carriage control.

c) To intermix SYS011 and SYS003 output:

```
//SYS011 ACCESS P1
```

(following the ALLOC card of example b)

or

```
//SYS011 ACCESS PX,SAME=SYS003
```

d) The card punch and read may be handled similarly, using the key words PUNCH and READER, e.g.

```
//SYS002 ALLOC DS2, PUNCH=

//SYS005 ALLOC NEWLU,READER=
```

VI REENTRANT SUBROUTINES

A later version of the system will include facilities for the sharing of subroutines coded in a reentrant manner between different users in order to conserve memory.

This chapter will include notes on which routines are reentrant and how they affect load-editing of multiphase programs.

Rules for writing and incorporating reentrant routines into the system will be found in the design notebook.

The four following page numbers are reserved for expansion of this chapter.

VII JOB CONTROL

The general concepts of job control embodied in 44PS have been retained (separately within each logical user) in MPS/44. Except as noted below, the format and use of each statement conform to those in 44 PS.

The MPS job control processor is loaded by the system to process each job or job step. It reads and interprets the input stream job control statements and provides the requested services and facilities. The job control statements describe the machine and system resources needed to execute a program.

When the job control processor finishes processing control statements it requests the system to load the loader-editor control program and returns control to the system. At the end of the job it provides the necessary end-of-job services.

The statements recognized by the job control processor are summarized in the Table 1, following, and described in detail directly following.

Table 1 - MPS/44 Job Control Statements

STATEMENT	FUNCTION
JOB	Defines start of a job.
EXEC	Defines start of a job step execution.
ALLOC	Allocates space for a new data set.
ACCESS	Permits access to an existing data set.
LABEL	Defines data set characteristics.
RESET	Restores unit assignments to standard accesses.
DELETE	Deletes a data set from a volume or a member from a directoried data set.
CONDENSE	Condenses a directoried data set.
REWIND	Rewinds a tape volume; repositions a data set on a direct access volume to beginning.
UNLOAD	Rewinds and unloads a tape; unloads a queue.
PAUSE	Same as 44PS PAUSE
RENAME	Rename data set or member.
*(comments)	Allows logging of comments to system log.
/*	Delimits the end of data in the input stream.
/&	Delimits the end of a job.

Continuation cards are allowed, and are in the same format as for 44PS.

JOB STATEMENT

The JOB statement defines the start of a job. The first JOB card of each logical user must immediately follow the 12-7-8-9 job stream delimiter card (see Chapter XI).

The JOB statement format is:

// [jobname] JOB [DUMP / NODUMP]

The description for the parameters of the JOB statement is given in the GUIDE TO SYSTEM USE. However, a jobname is necessary in order to identify to the operator the job(s) currently being executed by the system. This name should have 3 or more characters.

EXEC STATEMENT

The EXEC statement defines the end of job control information for a job step. It causes job control to request execution of the loader-editor which will then prepare the specified program for execution. One EXEC statement is required for each job step.

The EXEC statement format is:

// [stepname] EXEC [program] [(Parameter list)] [, (user program switch)]

The parameters are as follows:

stepname

This field specifies the name of the job step. If used, the step name must be one to eight alphameric characters, the first of which is alphabetic. A job step name is stored in bytes 32 through 39 of the logical user header. If a job terminates abnormally, this information can be used to help determine the point of termination.

program

This field specifies the name of an Absolute Relocatable Phase present in the MPS system library (SDSABS). The name must be one to eight alphanumeric characters, the first of which must be alphabetic. When a system program is to be executed, its name is given in this field. For problem programs in module form, this field must be omitted. In that case, the EXEC card will be followed by loader-editor control statements. Description of the loader-editor control statements is given in Chapter IX.

parameter list

This field may specify up to nine parameters to be used by the program to be executed. During execution of the job step, these parameters are saved in bytes 56 through 127 of the logical user header. A programmer can use them to establish his own execution-time options. Note that nine parameters are allowed, as compared to six in 44PS. Parameters in the field are separated by commas, and the entire field is enclosed in parentheses. Each parameter may consist of up to eight alphanumeric characters. Caution should be exercised to avoid duplicate use of KEYWORDS utilized by the system programs.

user program switch

The user program switch is byte 40 in the logical user header. It may be used by the problem program for communication between job steps. The EXEC statement may specify an initial setting for the switch. Supervisor calls are required to reset or read the switch. The switch is reset to zeros at the beginning of each job. This field contains up to eight characters. Each character in the statement must be either 0,1, or X. A 0 in the statement causes its corresponding bit to be set to 0. A 1 results in a 1 bit; and an X indicates that the corresponding bits is to be left as is. If fewer than eight characters are given, the rightmost positions are assumed to be X's.

ALLOC STATEMENT

The ALLOC statement is used to create and name a data set. The ALLOC stateme conforms in all respects to that of the 44PS except that the CATLG keyword is not permitted. In addition, selection of Queues or virtual unit record devices such as Readers, Punches, and Printers may be specified by placing any one of the following Keywords in the type field of the volume entry:QUEUE, READER, PUNCH, PRINTER.

It is possible for a single logical user to have several virtual printers assigned for his use for outputting different data in different formats. This process is detailed in Chapter V-A.

ACCESS STATEMENT

The ACCESS statement conforms in all respects to that of 44PS with the addition of the QUEUE, READER, PUNCH, and PRINTER Keywords. The use of these keywords is described under ALLOC. Note, however, that no new virtual device assignments will be made for the ACCESS statement. Units must be accessed previously ALLOCed printers (or punches) using the data set names from the ALLOC cards or the SAME=SYSXXX designation.

Since there is no system catalog, the first ACCESS (or ALLOC) for each data set in a job must have a device type or device address specified.

LABEL STATEMENT

The LABEL statement conforms in all respects to that of the 44PS except that the expiration date field must be omitted. All data sets are given the expiration date 99999 to enable such disks to be safely condensed with the 44PS SQUEEZE utility.

The following table shows the available system units, their hexadecimal codes (used in SVC parameters), and their normal functions. Each logical user has a separate unit table and independent use of any or all units. FORTRAN data set reference numbers have the same equivalences to system units as in 44PS (see Chapter XIII).

Table 2 - System Units

Word No. in System Unit Table	Hex Code (SYSUNI Pointer)	Name	Function
0	01	SYSAB1	Read system transients from disk
1	02	SYSAB2	Used by Loader Editor to write user phases to disk and to load them from di
2	03	SYSREL	System Module Library
3	04	SYSLOG	System/operator communication
4	05	SYSRDR	Job control input
5	06	SYSIPT	Normal (card) input
6	07	SYSLST	Job control output
7	08	SYSOPT	Normal (print) output
8	09	SYSPCH	Punch output
9	0A	SYSPSD	Current Module Directory
10	0B	SYSDMY	Internal system use
11	0C	SYSUAS	Job control unit assignment table
12	0D	SYSPRV	User Module Library
13	0E	SYSVDS	Virtual device and queue data set
14	0F		Reserved for system use
15	10	SYS000	Language processor module output*
16	11	SYS001	System work unit*
17	12	SYS002	Utility input*
18	13	SYS003	Utility output*
19	14	SYS004	Free Units
. . .	. . .	. . .	
54	37	SYS039	
55	38	SYS040	Reserved for system use
. . .	. . .	. . .	
62	3F	SYS047	
63	40	SYS048	User Program check device

*SYS000, SYS001, SYS002, and SYS003 are free for user use when not otherwise occupied.

VIII. FORTRAN COMPILER UNDER MPS

The FORTRAN compiler is essentially the 44PS compiler, modified internally to conform to the requirements of MPS, but functioning in all respects like the 44PS Compiler.

In order to minimize the amount of memory occupied by the compiler, the amount of space it uses for its internal tables and data storage is placed under control of the user by means of an additional EXEC option, COREnnnn. If this option is not specified, the compiler can compile a program of 100 to 200 statements. The value of nnnn is the number of additional pages of core (each page is 2048 bytes) to be added to the data storage areas. Thus CORE0004 adds another 8192 bytes. Since the exact size of the program which may be compiled depends on its own structure, it is not possible to predict what value of CORE is needed for a given program. If compilation terminates because of insufficient storage, this value must be empirically increased. Note that making it arbitrarily large without serious thought simply decreases the probability of having enough room in memory for the compiler. Since in general it is desirable to break large programs into subroutines of less than 100 statements, this uncertainty is not serious, and most programs should be able to be compiled without the CORE option.

The compiler may be freely multiprogrammed with other jobs, and in fact several copies of the compiler may be executing simultaneously.

The compiler's I/O units may sometimes be accessed to queues. See Chapter IX-C for details.

IX. THE LOADER-EDITOR

A. General Description

One significant departure of MPS/44 from the 44PS scheme is the dropping of the concept of the user phase which can be preserved from job to job or job step to job step. Rather the function of editing the individual subroutines and library routines into a program is performed every time it is loaded. This enables the program to be loaded into memory wherever space exists, which enables it to co-exist with other programs in a multiprogramming environment. In contrast, a phase usually occupies a specific place in memory and can only be loaded if that place is available. The concept of phase is preserved within an individual job step to make possible overlay jobs.

The component which performs this function is called the loader-editor. Its use is very similar to the use of the 44PS linkage editor, and information omitted below is available in the 44PS manuals.

The Loader-Editor performs 2 functions at program execution time:

1) Loading of system programs residing on the absolute phase library (SYSAB1)
2) Linking and loading of problem programs which can be input from:
   - module decks (SYSIPT)
   - module library (SYS000 with SYSPS Das directory)
   - private library (SYSPRV)
   - systems library (SYSREL)

The output is in form of absolute phases on the user's phase library (SYSAB2). This data set is cleaned out at the start of editing. Normally the first phase is loaded following editing. Approximately 50 phases may be accomodated in one job step. Since most jobs at this magnitude will be broken into separate logical users (each of which may have 50 phases) this should be adequate.

In two respects, the MPS/44 loader editor is more versatile than the 44PS linkage editor. First, it recognizes the private module library as an entity in addition to the system library and language processor output data set. Second, it recognizes re-entrant programs as such (described in Chapter VI).

To use a private module library, the private module library must be on a disk directoried data set blocked 360. Unit SYSPRV must be accessed to it. During autolinking, this data set is searched before the system module library.

Initially any module decks are copied onto SYS0000 and entered into the SYSPSD directory following the language processor output of previous job steps. Control cards are placed directly after the module decks, and in general are used in the same manner as the corresponding 44PS cards.

When load-editing programs consisting of modules, some or all of the following units must be accessed to data sets specified as shown. (None are required for loading of system phases.)

Unit	Block Size	Format	Medium
SYS001	360	sequential	queue, disk, or tape
SYSAB2	720	directoried	disk
SYSPSD	360	directoried	disk
SYS000	360	sequential	disk or tape
SYSPRV	360	directoried	disk

Access for SYS001, SYSAB2, SYS000, and SYSPSD are always required. SYS000 and SYSPSD may be accessed to IGN if all modules are on SYSPRV and SYSREL. An access for SYSPRV is only needed if this facility is used. In addition, a private library may be substituted for the system module library (SDSREL) by accessing SYSREL to it. This library must be a directoried disk data set blocked 360 and must contain any system modules needed for the job step in addition to the user's modules.

Programs are loaded into a contiguous area of memory. Multi-phase structures are constructed in a manner exactly analogous to their 44PS counterparts except that they are placed wherever space is available. Enough contiguous memory must be available to contain the specified structure. If not enough space is available, the job step is terminated. To minimize this possibility, global symbols (which are not contiguous to the body of the program) should be used for large arrays.

If the loader-editor is unable to return its core to the storage pool following a terminal error, the job is cancelled.

Errors: The standard 44PS error codes are used, with these additions and changes:

KA90I - Insufficient space for the combined linkage table and control dictionary. The control dictionary has 275 entries, unlike 44PS.

KA94I - Error in closing one or more I/O units.

KA96I - Insufficient core for user program or requested systems program.

KA97I - Errors in SVCs SET, FREE, or LOAD during loading of user programs.

KA98I - Errors in SVCs SET, FREE, or LOAD during loading of system programs.

If any needed access cards are missing, the job step terminates with an appropriate KA message (KA83I).

Input-output error messages include a symbol identifying the unit involved. These symbols may be looked up in table 2 (page 21).

B. Control Cards

The following control cards are used in the sequence shown:

MODULE	(optional)
module deck	(optional)
. . .	. . .
PHASE	(necessary)
INCLUDE	(necessary)
. . .	. . .
INCLUDE	(optional)
PHASE	(optional)
INCLUDE	(optional)
. . .	. . .
ENTRY	(optional)

Except as specified, all are the same as their 44PS counterparts. Note that at least one PHASE card is always required, along with INCLUDE module, L for all modules on SYS000 (module decks or language processor output in previous job steps).

The format of the EXEC card for a program to be edited from modules is

//stepname EXEC

No program name is used. No options exist. MAP output is always provided. AUTOLINK is in force unless NOAUTO is used on PHASE cards. The form // EXEC (options) may be also used. In this case the options are saved for use by the user program.

The formats of the load-edit control cards are the same as in 44PS except as noted:

MODULE — The format is the same as in 44PS

PHASE — Name, origin $[\{\frac{\text{AUTO}}{\text{NOAUTO}}\}]$

-options for the origin are the same as in 44PS (including the use of S and *) except that absolute addresses are relative to a relocation constant determined at load time.
-AUTO searches SYSPRV first, then SYSREL.

INCLUDE — module name, $\left\{\begin{matrix}\text{R}\\\text{L}\\\text{P}\end{matrix}\right\}$ [,(csname,. . .)]

-R indicates that module is on SYSREL

-L indicates that module is on SYS000
-P indicates that module is on SYSPRV

ENTRY — name
-same as 44PS

Examples of job decks are in Chapter XI.

C. Use of Queues

SYS001 may always be acessed to a queue.

Under certain restricted, but common conditions, SYS000 and SYSPSD may be accessed to queues (each to a separate data set).

SYS000 may be accessed to a queue provided that modules are written into it in exactly the order in which they are called for by INCLUDE cards.

SYSPSD may be accessed to a queue if there is only one module on SYS000. To set it up the following cards must be used prior to the first FORTRAN or LOAD edit step.

```
//SYSPSD ACCESS IGN
/*
//SYSPSD ALLOC  dsname, QUEUE=
//  LABEL  360
//  EXEC  PSDSET
```

The first access and /* are mandatory. Without them job control will attempt to read SYSPSD while attempting to initialize it during the processing of the EXEC card. Since the queue is empty at that point, job control will hang up waiting for a record to be written, which will never happen. (The program PSDSET writes this record.)

D. Writing Into Private Module Libraries

It is possible to have the FORTRAN compiler or loader-editor write modules directly into a private module library. Two procedures are available:

If multipile member names are required, use the following:

```
//SYSPSD ACCESS any ds
//SYS000 ACCESS private(name 1,name2,. . .),NEW
//NAME  EXEC  FORTRAN
```

This should be placed in a job by itself. This procedure is also safer than the one described below and should be used where possible.

If no such new member needs multiple names, the following procedure can be used to write many new members in one job. The loader-editor may also be used to write module decks into the user's data set

```
//SYSPSD ACCESS IGN
/*
//SYSPSD ACCESS mydataset
//SYS000 ACCESS mydataset,EXT
//NAME  EXEC  FORTRAN
      etc. (more job steps)
```

If the data set is empty at the start, a dummy member must first be entered with the DSINIT program.

The preceding access and /* is necessary to prevent job control from clearing the SYSPSD directory, which would wipe out the user's data set.

If the user's data set is to be cleared first, he should use DSINIT prior to any compilation or load-edits.

X. LOGICAL USER AND JOB DECK PROCESSING

Initially the entire logical user deck is read into a virtual reader queue. No processing occurs until the end-of-deck (12-7-8-9) card is detected.

Following this, priority is assigned, the necessary control blocks are set up, and job control begins processing the first job. In general, job processing is very similar to 44PS, except in certain significant details which will be explained below.

Once processing begins, the logical user is referred to either by the 2 digit ID code typed by the system at the start, or by the name on the JOB card for the current job.

The "pseudo-directory" data set (accessed to SYSPSD as a directory to the modules on SYS000) is initialized at the first /*, EXEC, or PAUSE following the first SYSPSD access card. This difference from 44PS (in which the JOB card processor performs this function) affects the use of private module libraries as is described in Chapter IX.

All programs (including the compiler, utilities, other self-relocating programs, and user programs edited for modules) are loaded by the loader editor. When the requested program is set into execution, general register 15 contains the entry address, and register 13 contains the address of a FORTRAN-type save area in the user program zone which may be used either as a work area or a register storage area. This is the same area whose address is placed in the logical user leader. Note that the program entered by the loader-editor need not save the registers unless its own internal workings require it. In addition, a virtual device buffer is also set up in the program zone.

The first SVC FREE done by the user program in the program zone will not necessarily return a bead aligned on a doubleword.

The FORTRAN compiler writes its output on SYS000 (using SYSPSD as a directory) where it is later processed by the loader-editor.

The edited program is entered directly from the loader-editor as part of the same job step.

Following the first job, any following jobs in the logical user are processed in the same way, exactly analogous to a 44PS batch stream.

Set up and processing of one user is multiprogrammed with execution of any and all older users; however job control and the loader-editor can only work on one user at a time. Therefore, if any user requires one of these services while the service is busy, he waits until it completes its current job (e.g. until job control completes the processing of the next EXEC card and turns control over to that program).

If a user requests facilities which are unavailable, it is disabled (status 18). The operator must enable it manually when the needed facilities become available.

XI. SUMMARY OF INPUT DECK STRUCTURES

A. Logical User Deck Definition

The deck for each logical user consists of a job or jobs in the usual 44PS card format, headed by a card containing

{12, 7, 8, 9} punches in col. 1, followed by a user name of 8 or fewer characters, left-justified in col. 2-9,

and ended by a card containing {12, 7, 8, 9} punches in col. 1 and 2.

In the following examples, these cards will be denoted by ¢ (pronounced "bar-nine") and ¢¢ respectively. Note that the combination 12-7-8-9 can be produced by multipunching 9 and | (vertical bar).

B. General Comments; Essential Data Sets

The use of a name on the JOB card is strongly recommended in the case of logical users with more than 1 job, since it is the only way to follow the progress of this succession of jobs. The name must have 3 or more characters.

The use of step names on all // EXEC cards is strongly recommended to enable the progress of the job to be followed. Remember that printing of output does not begin until a logical user is deleted.

For jobs involving editing programs from modules by the loader-editor, ACCESS cards must always be provided for the following units: SYSAB2, SYSPSD, SYS000, SYS001. The user may supply any suitable data sets, subject to the following consideration:

SYSAB2 must access a disk directoried data set with 720 byte blocks, large enough to contain the user's entire program after LOAD-EDIT. It is cleared by the loader-editor before writing the new phases.

SYSPSD must access a disk directoried data set with 360 byte blocks. It is used to hold a directory to the user's modules. There must be room for 1 entry (24 bytes) for each module deck and each preceding FORTRAN compilation or assembly, plus 1 additional entry. If there is only one module on SYS000, a queue may be used for SYSPSD by initializing it first, as described in Chapter IX-C.

SYS000 should access a tape or disk data set with 360 byte blocks. It must have one block for each 5 module deck cards or equivalent language processor output; if the modules are on SYS000 in the order required by the loader-editor, and each is used only once, a queue may be used.

SYS001 can always access a queue for the purposes of the compiler and loader-editor.

If all modules are in the system module library or in private libraries accessed by SYSREL and SYSPRV, SYS000 and SYSPSD may be accessed to IGN.

C. Deck Examples

1. Execution of a system program, such as COPIER

```
$USERNAME
//MYJOB JOB NODUMP
//SYS002 ACCESS input
//SYS003 ACCESS output
//STEP EXEC COPIER(options)
/*
/&
$$
```

2. To link and load a single module deck, one of the following sequences may be used:

a)
```
$ME2
//NAME JOB NODUMP
//SYSPSD ACCESS IGN
/*
//SYSPSD ALLOC DIR,QUEUE=
//  LABEL 360
//  EXEC PSDSET
//SYS001 ALLOC TEMP,QUEUE=
//  LABEL 360
//SYS000 ALLOC MOD,QUEUE=
//  LABEL 360
//SYSAB2 ACCESS PH,OCO='MPSWRK'
//ABC EXEC
  MODULE A
  Module cards
  PHASE A,S
  INCLUDE A,L
/*
data cards for A
/*
/&
$$
```


b)
```
$MYNAME
//NAME JOB DUMP
//SYS001 ALLOC TEMP,QUEUE=
//  LABEL 360
//SYSAB2 ACCESS MYPHAS,OC1='MPSWRK'
//SYS000 ACCESS SCRATCH,OC1='MPSWRK'
//SYSPSD ACCESS MYDREC,OC1='MPSWRK'
//ME EXEC
  MODULE A
  Module Cards
  PHASE A,*,NOAUTO
  INCLUDE A,L
/*
Any Data Cards
/&
$$
```

3. To link and load several FORTRAN routines into one phase:

```
$USER
//ABC JOB
//SYSAB2 ACCESS MYPHAS,OC1='MPSWRK'
//SYS001 ALLOC TEMP,QUEUE=
//  LABEL 360
//SYS000 ACCESS MYDATA,OC1='MPSWRK'
//SYSPSD ACCESS MYDREC,OC1='MPSWRK'
//P1 EXEC FORTRAN
FORTRAN Program
/*
//P2 EXEC FORTRAN(MAP)
. . .
/*
//P3 EXEC FORTRAN(CORE002,MAP)
. . .
/*
. . .
//P5 EXEC
  MODULE P4 (from previous compilations)
. . .
  PHASE MYPROG,*
  INCLUDE P1,L
  INCLUDE P2,L
  INCLUDE P3,L
  INCLUDE P4,L
/*
  data for MYPROG
/*
/&
$$
```

Note that since the INCLUDE cards are in the same order as the compilations, a queue could be used for SYS000.

4. To load and link modules contained only in the system module library

```
$NAME
//XX JOB NODUMP
//SYS000 ACCESS IGN
//SYSPSD ACCESS IGN
//SYS001 ALLOC TEMP,QUEUE=
//   LABEL 360
//SYSAB2 ACCESS MYPHAS,0C1='MPSWRK'
//   EXEC
  PHASE EBCDIC,*
  INCLUDE EBCDIC,R
/*
  input cards
/&
$$
```


XII. OPERATOR-SYSTEM COMMUNICATIONS

A. Use of Console Typewriter in User Programs

The console typewriter may be used for user input and output.

From the viewpoint of the programmer, it is used in exactly the same manner as 44PS, except that all input is normally converted automatically to upper case. Because of the details of the character codes used, the carriage return is also converted to 55 (Hex). None of the digits or special characters is affected. Assembly language programs (or FORTRAN programs directly calling the SVCs) have the option of suppressing the case conversion.

From the viewpoint of the operator, the typewriter is manipulated in a manner different from 44PS. The differences arise primarily from the fact that many users may be doing input and output intermixed. On output, the system prefixes the logical user identification code to the output line in the form:

nn>usermessage

where nn is the logical user's ID code in hexadecimal (obtained from the typeout during the job initiation).

When the program issues a READ, the typewriter types nn? where nn is the logical user ID code for the program which issued the READ. The PROCD light does not stay on. When the operator wishes to input the requested data, he must hit REQUEST. When the PROCD light comes on, he types the logical user ID followed by an EOB. The carriage then returns and the symbol < is typed and the PROCD light comes on, following which the user should enter his message.

The cancel key is used for error correction in the same manner as in 44PS.

If an attempt is made to enter a message to a user which is not waiting for input, the system simply types a *, and the keyboard remains locked.

Several users may be waiting for input at the same time. Their input may be entered in any order.

As was noted above, there is no programming difference: The user ID code is not transmitted to the user's program; only the data are transmitted, exactly as in 44PS.

While the program is waiting for input, another user will receive the use of the CPU. In the case of FORTRAN I/O, there is no option. Assembly language programs (and FORTRAN programs directly calling the SVCs) may of course continue to compute following the SVC READ, and issue the SVC CHECK only at the point where they cannot go on without the requested input. It is permissible to test the post request flag (byte 4) of the RCB (request control block) and only issue the CHECK after the operation is complete. This makes it possible to set up a situation where the program has two alternate courses of action depending on whether or not the input has occurred. The CHECK must, however, be issued before the input data are used or the RCB reused.

It is also permissible to do READs on separate RCBs, thus having several outstanding READs from the same logical user. In this situation, the first operator input goes to the most recent read, the next to the next most recent, etc. Before using the data, a CHECK should be issued for the appropriate RCB.

Assembly language programs which issue an SVC READ and wish to withdraw it before the operator responds, may do so by issuing a SVC REWIND to the typewriter. This clears all pending reads for the logical user.

For programs which request only an EOB from the operator, he must type the logical user ID, followed by EOB, followed by the requested EOB. This includes FORTRAN PAUSE statements.

B. Operator Intervention Program

The Operator Intervention Program is loaded by the Job Scheduler immediately following IPL (initial program loading) completion and never exits. It accepts requests from the operator to perform a variety of functions which include listing the current users, altering user priorities, cancelling users, and others.

The facilities of the operator are requested by typing 01 followed by an EOB, and then a command. (01 is the logical user identification of the operator).

Commands for the Operator resemble assembler instructions in that they have a name field, an operation field, and an operand field separated by blank spaces. The name field, when present, usually specifies a user ID or job name, while the operand field interpretation varies with the operation. Commands are free-form as for the assembler (that is, the first occurrence of a blank after the start of any field terminates the field, and blanks between fields are not significant). The available commands, which are discussed below are:

LIST	List Users
IBALL	Give Debug Dumps
DISABLE	Stall a User
ENABLE	Restart a User
CANCEL	Cancel a Job
DELETE	Delete a User
CALL	Load a System User
UNDER	Change Priorities

(Note: Only the first 4 letters of a command need be typed.)

In the following descriptions, the term LUSPEC is used to mean either a hexadecimal ID code (typed by the system when the user is first accepted) or a job name. The brackets follow the usual conventions described in the 44PS literature.

1. LIST - list users

Syntax: [LUSPEC] LIST

This command causes vital statistics of one or more users to be listed. If LUSPEC is given, the statistics for that user are given. Otherwise, all current

users are listed. No operand field is required. The statistics given (in order) are:

Priority Jobname User ID Elapsed CPUTime (sec.) Current Status

The status code indicates the current stage of processing. It is the sum of a basic status plus a modifier.

Basic status: (Hex) 00-User program
04-Job control
08-Loader editor
0C-User dump
10-User deletion
14-EOJS or CANCEL being processed
18-Disabled
1C-//PAUSE in effect

Modifiers (Hex): 0-in-service list - ready for specified stage
1-started executing, but waiting for I/O or some event
2-active, i.e. specified stage is executing or has been interrupted.

Examples: 0A (=08+2) means loader-editor is executing

01 (=00+1) means user program has started executing and is now waiting for I/O.

The elapsed time listed under MPS44 (logical user 00) is the total idle time since IPL. To determine the fractional time available to new users, take two LISTS a short time apart. The fractional idle time is the change in elapsed time for MPS44 divided by the elapsed real time between the LISTS.

2. IBALL - Provide debug information

Syntax: [C] IBALL { {L, P} hexnum | {H, U} hexid [, modifier] }

This command is used for debugging purposes. It causes a selective core dump whose format and extent is determined by the name and operation fields. The dump is always in hexadecimal unless "C" is specified in the name field, in which case the dump is in character (EBCDIC) form. Each line is preceded by the address (in hex) of the first byte. The first character of the operand field determines the form of the dump specification as follows:

L = loc. given by "hexnum"
P = loc. given by bead pointer "hexnum"
H = logical User Header specified by "hexid"
U = unit table for user specified by "hexid"

If we let "A" denote the effective address as determined from either the L or P specification, then the dump for these specifications always extends from A to A+31 (i.e. 8 full words or 1 typewriter line). For a U specification, the unit table is dumped, one unit per line, from the LUH associated with user ID "hexid". For an H specification, words 2-13 and 44-56 of the associated LUH are dumped unless the modifier is present, in which case 8 full words are dumped beginning at the word number (decimal) given by the modifier. (See design notebook for word numbers in the LUH.)

For example, the following command dumps words 8-15 of the logical user head of user 3D. The first 8 characters typed are the step name for the current job step.

C IBALL H3D,8

3. DISABLE - stall a non-system user

Syntax: {LUSPEC} DISABLE

The user specified by LUSPEC is placed in the service list with a code of X'18'. This action effectively suspends processing indefinitely for this user. If the request is to disable a system user, it is ignored and a message is printed.

4. ENABLE - restart a stalled user

Syntax: {LUSPEC} ENABLE

The user specified by LUS PEC must currently be in the service list with a code of X'18'. The execution of this command causes him to be restored to his status at the time he was disabled. If the user has not been disabled, the request is ignored and a message is printed.

5. CANCEL - terminate a job

Syntax: {LUSPEC} CANCEL [DUMP]

The current job for the user specified by LUSPEC is canceled. If DUMP is specified or if the DUMP option was given on the JOB card, a memory dump is written before the cancelation occurs. Note that system users may not be cancelled and a request to do so will be ignored. Note that this cancels a job and not a logical user. If a logical user contains several jobs, the next job will be

executed at the current priority level. User status must be less than 0C in order to cancel.

6. DELETE - delete a non-system user

Syntax: {LUSPEC} DELETE [DUMP]

Processing occurs as if a CANCEL request had been issued, except that upon completion of the cancellation procedure, the logical user specified by LUSPEC is deleted.

7. CALL - invoke a system user

Syntax: CALL {username}

where "username" is a 1-8 character name denoting the system user to be loaded.

The request bit is set in the function/user table for the system user specified, causing the job scheduler to load the user as soon as possible. Invalid user names are ignored. Descriptions of the system users which may be called are given elsewhere. As example is the physical user setup program.

8. UNDER - alter priorities

Syntax: {LUSPEC1} UNDER {LUSPEC2}

The user specified by LUSPEC1 is removed from his current position in the priority list and inserted immediately following the user LUSPEC2. It is impossible, using this command, to move a user to the top of the list.

Table 3 - Operator Program Error Codes

Number (hex)	Error
1	Name field exceeds 8 chars.
2	No such operation
3	Required name field not present
4	No such user
5	Invalid operation (name too long)
6	Error loading transient rtne.
A	Can't find user that should be in priority list
14	Bead pointer exceeds core size
15	No such user ID
16	Modifier too big
17	Code letter in error
1E	System user specified
20	Invalid user status
28	User name too long
29	User name missing
2A	User name not in function/user table
2F	LUSPEC1=LUSPEC2
30	User missing from list
32	System user specified
33	User status invalid

C. Physical User Setup Program

Physical User Setup is accomplished through a system user called the Physical User Setup Program. The services of this program are requested with the operator command CALL PHYSICAL.

When the above command has been entered, the following is typed:

```
04>ENTER PHYSICAL USER PARAMETERS
04?
```

The operator then responds by typing

```
04 (EOB)
```

and either of the following lines:

```
Name [,blocksize] [,nsyms]
Name, NOSYM
```

Name is the name of a new physical user, or the name of an existing one to which additional logical users are to be added.

The following parameters are ignored if a global symbol zone already exists for this physical user which was set up previously.

NOSYM means omit the global symbols. If NOSYM is omitted, a global symbol zone will be set up for a new physical user.

blocksize is the size of the primary memory block at the global symbol zone. It should be given in pages (1 page = 2048 bytes). If omitted, 1 page is assumed.

nsyms is the number of entries to be provided for in the initial section of the symbol table. If omitted, 32 is assumed. This is not an absolute limit on the number of symbols, but memory allocation is most efficient if enough memory for the entire symbol table is allocated at the start.

After these data have been entered, the following message appears:

```
04>ENTER LOGICAL USER NAMES
04?
```

The operator then types the list of logical user names to be added to this physical user. The list is a string of names separated by commas. If more names are to be entered than fit on one line, the P.U. setup program must be called again to add the rest.

The names are interpreted as follows: If there is a current job in the system with this name (name field on the JOB card) it is assumed to refer to that logical user. In such a case, the referenced job must be waiting at a job control pause (// PAUSE) with no zones set. If there is no such job in the system, the name is taken to be the name of a logical user which has not yet been started. In this case, the name will be the name on the 12-7-8-9 header card.

The reason that current users must be at a job control pause with no zones set is that setting up a physical user involves reassigning protection keys, which is difficult to do if the user is occupying any storage.

There are thus two procedures for creating a new physical user or adding new logical users to it:

1. CALL PHYSICAL before entering the logical users into the system.

2. Enter the logical users into the system, with a // PAUSE card in each deck prior to any programs which set zones. Then when all have reached the // PAUSE, CALL PHYSICAL. Note that in this case the job cards of the jobs containing the // PAUSE must have unique names. Following setup the physical user, enter an EOB to each logical user to restart it.

D. Virtual Device Utility Control

Pages are reserved for a description of the operator controls over the virtual devices to be inserted when the facility is implemented.

XIII. FORTRAN LIBRARY IN MPS/44

A. Introduction

The MPS FORTRAN library is exactly the same as the 44PS library except for details described in the next section. Later versions of the system will incorporate a re-entrant FORTRAN library, which will result in more efficient use of storage.

B. Changes and Additions to Specifications of the FORTRAN Library Facilities

1. Data Set Reference Numbers 0-15 are available. The equivalences to SYSxxx symbols are as follows:

0 - SYSLOG	4 - SYS004	8 - SYS000	12 - SYS008
1 - SYS001	5 - SYSIPT	9 - SYS005	13 - SYS009
2 - SYS002	6 - SYSOPT	10 - SYS006	14 - SYS010
3 - SYS003	7 - SYSPCH	11 - SYS007	15 - SYS011

Note that unit 0 is available for writing and reading the console typewriter without using an ACCESS card.

ACCESS cards are required for all units but 0, 5, 6, and 7. As usual, access cards may be used for units 5, 6, and 7 if it is desired for these units to access non-standard data sets.

2. To retart the program after a PAUSE statement, the operator must hit REQUEST, type the logical user ID, then EOB, and then another EOB.

3. In multiphase jobs, DEFINE FILE does not have to be in the root phase, although it should be if compatibility with 44PS is desired.

4. If the data set size specification in a DEFINE FILE statement is larger than the size of the accessed data set (but no attempt is made to go beyond the actual end of the data set), the message 0A2171 will be printed at the end of the job. This may be ignored.

5. A new I/O error code has been added: 0A2371 means I/O error return from SVC POINT during a direct access I/O statement execution. This actually can occur in 44PS also, but the message does not appear in the manual.

6. An additional format code, R conversion, is available. This is an alphameric conversion identical to A conversion except that with R conversion, the characters are right-justified and padded at the left with leading zeroes. This code may only be used in object-time format statements since although the format statement processor recognizes it, the compiler does not.

7. In addition to the 0A-type messages, all FORTRAN jobs end with an MPS end code. The following end codes are defined:

X000000Y -

X = 8 if a job control card has been read and saved by the FORTRAN I/O routines. X = 0 otherwise.

Y is the highest severity in the job step.

00020101 -

There is no room in memory for an I/O buffer.

8. When the statement CALL LINK is executed, all units are repositioned (rewound) except SYSOPT, SYSIPT, and SYSPCH. If this is not desired, a root phase structure (CALL LOAD) must be used. This is true of 44PS as well, although it is not stated in the manuals.

9. In the initial versions of the system, all program check interrupts are fatal. This will later be modified to conform to 44PS standards.

XIV. UTILITY PROGRAMS

A. Sequential Copy Program

1. General description

COPIER is a general purpose utility program which copies from any sequential data set to any other. Subject to the limitations imposed by the output device, it can copy variable length multi-file data. If the output unit has a fixed block size, action (i.e. truncation or padding) depends on the characteristics of the device. Incorrect length indication is always suppressed.

Members of data sets may be copied by accessing the members rather than the whole data set. In such cases, the output member will, of course, be founded off to an integral number of blocks.

There are various options, requested with EXEC parameters. Some are specific to certain types of devices (e.g. multifile options are only valid for magnetic tapes), and if used with inappropriate devices, action depends on the device or the READ/WRITE handler's action. There are twelve different options, of which a maximum of nine may be specified at one time. This should be adequate for all operations.

Options include buffer size, record skipping, file skipping, copying several files, copying a fixed number of records, testing for valid printer carriage control characters, end-of-deck symbol for card input, open and close options, blocking and deblocking, and hexadecimal output.

Output is always a single file, even when multi-file input is used.

If blocking and deblocking are not in force, the output block size equals the input block size, and may be less than the buffer size. If blocking or deblocking is in force, the output block size is fixed, as specified by the EXEC parameters, and padded at the right with blanks. If hexadecimal output is requested, the output block size is twice that implied by the EXEC parameters.

If storage is not available for buffers, a message is written to the operator giving the error code and instructing him to signal EOB to terminate or enter 1 to retry. If he is able to make storage available to the program by deleting another user (or waiting until one terminates by itself) he may do so and then enter 1.

2. Use of the program

The job step consists of three cards:

```
//SYS002  ACCESS  input
//SYS003  ACCESS  output
//  EXEC    COPIER[(options)]
```

Options may be listed in any order. Descriptions of the options follow. No options need be specified if none are needed.

In the following, a prototype of each option is stated, followed by its description. Upper case letters must be copied exactly; lower case letters represent variable fields. Leading zeroes need not be entered.

a) SIZEnnnn - The input buffer is nnnn bytes. If this option is omitted, SIZE720 is assumed. If no blocking is in force, this represents the largest possible input block. If blocking is in force, the input block size is nnnn divided by the blocking factor.

b) SKIPnnnn - Skip the first nnnn output blocks (i.e. do not output them). Output blocks are counted following any blocking and deblocking operations. If this option is omitted, no skipping occurs.

c) FILEnnnn - Copy nnnn successive files. If omitted, copy one file. The output data set always contains one file with all the data. If nnnn is specified as zero (FILE0), nothing will be processed. This may be used with SKPF to simply position a tape at a specified file. This option is valid for tape input only.

d) COPYnnnn - Copy the nnnn blocks following any which were skipped. Output blocks are counted, following any blocking and deblocking operations. If this option is omitted, processing continues for a specified number of files.

NOTE: The operation ends as soon as the first of the SKIP, FILE, COPY, and END@ options is satisfied.

e) SKPFnnnn - Skip nnnn files prior to any other processing called for by other options. If this option is omitted, no files are skipped.

f) CARR - Check byte 1 of each output block. Replace it by blank if it is not a valid carriage control character. If the option is omitted, byte 1 is not tested.

NOTE: For printing, in order to actually obtain carriage control, it is necessary to place the following card directly after the SYS003 access card:

// LABEL ,CTLASA

g) END@xxxx - Mark end of a card input deck. The xxxx are any four or fewer characters. Processing stops when an input block containing these characters in col. 1-4 is encountered. If fewer than four characters are specified, blanks are assumed at the right. The xxxx block is not copied into the output data set. The test is made on all blocks including those being skipped by the SKIP option, but not on files skipped by the SKPF option. If blocking is in force, the last output block may be short, as it will contain only those input blocks read prior to the xxxx record, and will not be padded beyond that. If the option is omitted, no test is made.

This option is primarily for use with card input since the only end-of-file indication on the card reader is the end of the logical user deck.

h) Data set open options: The standard action is to open both SYS002 and SYS003 with repositioning. To open without repositioning, specify OPEN2, OPEN3, OPEN23, or OPEN32. 23 and 32 mean "both".

i) Data set close options: The standard action is to close both units without reposition. To alter this, specify CLOSxy, where x is a close option for SYS002, and y is a close option for SYS003. Options must be given for both units if needed for either. x and y may have the following values:

S - Close without reposition
R - Close with reposition
U - Close and disconnect (unload tape)
N - Don't close at all (applicable to SYS002 only).

j) Blocking and deblocking options -

BLOKmmmm
DBLKnnnn

If both are omitted, the output block size equals the input block size. mmmm is the number of input blocks to be combined. nnnn specifies the number of output blocks into which the buffer is to be divided. Both options may be specified in order to obtain output blocks which are not multiples of the input block size or vice versa. In this case mmmm input blocks are first combined and then divided into nnnn output blocks. The program assumes that the input block size is equal to the buffer size divided by mmmm, and the output block size is equal to the buffer

size divided by nnnn, regardless of the actual input block size. Therefore the SIZE parameter must always be stated exactly when blocking or deblocking is in force. An error termination occurs if either mmmm or nnnn does not evenly divide the buffer size. If input blocks are short, they will be padded with blanks, but, in the case of deblocking, only enough blocks will be output to completely contain the input block. If blocking is in force, and multiple files are being copied, a given block may contain records from the end of one file and the beginning of the next. If BLOK or DBLK is used, the output blocks are always the same length, padded with blanks if necessary. The specification BLOK1 or DBLK1 may be used to pad records at the right without blocking or deblocking.

k) HEX - Output in hexadecimal. This also causes the output block size to be double that inferred from the SIZE, BLOK, and DBLK options. The user must take care that this fits into the maximum block size of the output unit. CARR will be ignored if present, and // LABEL ,CTLASA should not be used.

NOTE: Illegal characters in numeric fields are ignored. For example: SIZE88M9 is taken as SIZE889.

3. Error conditions

Certain conditions result in immediate termination with the message "COPIER ERR. nnnnnnnn" printed on SYSLST. If the units have already been opened, they are first closed. Errors when closing units after an open error are ignored.

Following are the values of nnnnnnnn:

0000nnrr - error in SVC CLOSE. nn and rr are the error returns for SYS002 and SYS003 respectively (byte 3 of the close parameter).

02mmnnrr
03mmnnrr - I/O error mm on SYS002 or SYS003 respectively. If this was followed by a close error, the close error codes are given by nn and rr. If no close error, nn and rr are zero.

NOTE: error code 0204nnrr (EOF on SYS002) is issued only if COPY was specified but the specified number of files was reached before the specified number

of records. EOF on SYS002 is never an error if COPY is not specified.

0404nnrr - SVC OPEN error. nn and rr are the return codes for SYS002 and SYS003 respectively.

05mm0000 - Error return mm from SVC SETEXT when allocating space for buffers (or from SVC FREE when not preceded by SVC SETEXT). This message is issued only when the operator elects to terminate rather than retry the operation.

06000000 - The buffer size is not a multiple of the blocking or deblocking factor.

07000000 - Unrecognized EXEC parameter or unrecognized OPEN or CLOSE option.

08mm0000 - Error return mm from SVC FREE after successful SVC SETEXT.

4. Examples of Use

a)
```
//SYS002 ACCESS T,180=
//SYS003 ACCESS V,181=
//  EXEC  COPIER
```

This transfers all blocks from SYS002 to SYS003. Any that are longer than 720 bytes are truncated to 720 bytes. If SYS003 is accessed to SDSOPT, the first 120 characters of each block are printed; characters beyond this are lost.

b)
```
//SYS002 ACCESS T,180=
//SYS003 ACCESS X,181=
//  EXEC COPIER(DBLK6)
```

Each input block (up to 720 characters) is broken into 120 byte blocks, with the last block padded, if necessary, with blanks.

c)
```
//SYS002 ACCESS TAPE,180=
//SYS003 ACCESS SDSOPT
// LABEL ,CTLASA
// EXEC COPIER(DBLK6,CARR)
```

This functions exactly as example b, except that the first byte of each output block is blanked if it is not a valid carriage control character. The output is printed with carriage control.

d)
```
//SYS002 ACCESS SDSIPT
//SYS003 ACCESS SDSOPT
// EXEC COPIER(SIZE80,DBLK2,END@##)
```

This prints cards, with col. 1-40 on the first line, and col. 41-80 on the next line. No carriage control is used. The operation stops when a card is encountered with ## in col. 1-2, and blanks in col. 3-4.

e)
```
//SYS002 ACCESS SDSIPT
//SYS003 ACCESS DIRDS(MEMBER),OCO='DISKKK',NEW
//  EXEC COPIER(SIZE360,BLOK5,END@&&&&)
```

This may be used to copy a module deck into a member of a private module library. Note that the SIZE and BLOK options together provide for reading only col. 1-72 of each card.

f)
```
//SYS002 ACCESS SDSREL(SDSREL)
//SYS003 ACCESS SDSOPT
//  EXEC COPIER(SIZE360,DBLK15,COPY75)
//  EXEC COPIER(SIZE360,DBLK15,COPY75,HEX)
```

This lists the names of the first 75 members of SDSREL, first in text, then in hexadecimal.

g)
```
//SYS002 ACCESS TAPE,180=
//SYS003 ACCESS T,181=
//  EXEC  COPIER(SKPF5,FILE2,CLOSNS)
//  EXEC  COPIER(SKPF2,FILE3,CLOSRS,OPEN23)
//  EXEC  COPIER(FILE1,OPEN23,CLOSUU)
```

This copies from SYS002, files 6,7,10,11,12, and 1 in that order. Both tapes are unloaded at the end. Note that since SYS002 was not closed in the first step, the open in the second step will actually be ignored.

B. Data Set Initialization Program

1. Introduction

In multiuser systems having limited direct access storage capacity, it is difficult, or impossible, for individuals to use private disk packs since all drives must be available for shared use. Disk packs may not be changed if any users are present. Furthermore, the single disk storage drive, in particular, does not have enough capacity to allow for allocation of private data sets.

In order for users in such a system to make use of private module libraries or other directoried data sets, it is therefore necessary for such libraries to be stored on magnetic tape and rolled onto any available work data set for use. This requires a utility program to prepare such data sets for first use and perform related functions. It should be noted that it is not desirable to delete and allocate data sets on a shared disk since it is not possible to perform a SQUEEZE while any users are active. Space would thus become too fragmented, and in addition it would be difficult to determine the instantaneous data set configuration on the disk.

The main items of preparation consist of setting up a valid directory, clearing out any old data, and formatting the rest of the data set by filling it with binary zeroes.

The directory consists of a series of 24 byte entries describing the sizes and locations of the members of the data set. The first entry in the directory describes the directory itself in such a manner that for most purposes the directory may be treated as a member. The name in the directory descriptor is normally the same as the name of the data set as recorded in the disk's volume table of contents (VTOC).

There are three major (and common) situations which must be dealt with: 1. The data set is in totally unknown condition, and its directory is therefore a priori invalid. 2. The data set has been filled from tape, and the name in the directory descriptor does not match the name in the VTOC. 3. A data set directory is valid but the data set is empty when needed for access by SYS000 and SYSPSD (in which case it is first necessary to insert a member by some other means).

If the data set is already a valid directoried data set, it may be reinitialized by using the information in the directory descriptor. If, however, it cannot be assumed valid, then the user must supply all the information necessary to construct the directory. Since the initialization program is a user program, it does not have access to the name of the data set which its input unit is accessed to. The user must therefore supply this information as an input parameter.

If the data set has been copied from a tape, and the name in the directory descriptor does not match the name in the VTOC, it is necessary to alter the directory descriptor if job control is to be used to access individual members. This is not necessary if only the entire data set is accessed.

If the data set directory is valid but there are no members present, it may not be used by the language processors and loader-editor as a combined SYS000/SYSPSD data set since these programs assume that if there are no members, they should begin writing at block 1 of SYS000. In a combined data set, this would destroy the directory. A facility is therefore needed for easily inserting a first member prior to this use.

It should be noted that in principle such a data set must have been originally allocated with the FMT parameter. All public data sets must therefore be allocated in this manner.

The DSINIT program provides facilities for performing the functions described in the foregoing.

2. Use of the program

There are three main modes of operation:

a) RENAME - The name in the directory descriptor is altered but the data set is not otherwise altered.

b) REUSE - The data set is assumed to contain a valid directory. It is cleared out and reformatted with the same directory size and name. Optionally a dummy first member may be inserted.

c) If neither of the above modes is specified, the directory is assumed invalid and the user must supply all information required for constructing the directory. Optionally, a dummy first member may be inserted.

The program may be used with any disk data set blocked 360 bytes per block. It reads and writes on unit SYS003, which must be accessed to the data set.

There are three possible forms for the EXEC statement:

a) // EXEC DSINIT(dsname[,DIRCxxxx][DMEMyyyy])

b) // EXEC DSINIT(REUSE[,DMEMyyyy])

c) // EXEC DSINIT(RENAME,dsname)

The parameters may be listed in any order.

The first form is for complete respecification of the directory; the second, for reinitialization of a data set with a valid directory; the third, for simply replacing the name in the directory descriptor.

Parameter descriptions follow:

a) dsname - The name to be inserted into the directory descriptor. Normally it is the same as the name of the data set itself. If omitted, the job is cancelled (except in REUSE mode).

b) DIRCxxxx - xxxx is the number of entries desired in the directory (except for the directory descriptor, which is added by the program). If omitted, 14 is assumed. Note: the first block of the data set holds 14 directory entries; subsequent blocks hold 15. If xxxx is such that the last block of the directory is not filled, it is rounded upward. For example, DIRC15 actually produces a directory with room for 29 members. Illegal characters within xxxx are ignored; e.g. 25x3 is treated as 253.

c) DMEMyyyy - A first member is inserted, with the name yyyy. If less than four characters are supplied, it is padded with blanks at the right. The member itself will be one block long, and filled with binary zeroes. If this option is omitted, no dummy member is set up.

d) REUSE - Specifies that the name and directory size recorded in the existing directory should be used in reinitializing. If this directory is found to be invalid, the job is cancelled.

e) RENAME - Specifies that the program is simply to replace the name in

the directory descriptor with the supplied dsname.

Notes on the parameters - Parameters invalid in REUSE or RENAME mode will be ignored if supplied. A full eight characters are allowed for the data set name parameter (dsname). In order to make this possible, it is recognized by being the first parameter encountered which is not one of the keywords described above. Therefore a misspelled parameter will be used as the name, with no error indication given. Furthermore the names DIRC, DMEM, REUSE, and RENAME may of course not be used for data set names, or for the first characters of longer names.

The data set is closed as an output unit with repositioning unless either RENAME is invoked or errors have occurred. In these cases, it is closed as an input unit with repositioning.

3. Error conditions

Error conditions result in job cancellation. The CANCEL code gives the reason, as follows:

000008nn - I/O error code nn after SVC POINT.
00000Ann - I/O error code nn while writing directory.
00000Bnn - I/O error corde nn while zeroing out data set.
00000Cnn - I/O error code nn while reading directory descriptor.
00000D00 - REUSE was specified but directory format is invalid.
00000E00 - SYS003 Open error.
00000F0F - No data set name supplied although REUSE not specified.
0000F000 - SYS003 Close error.
0000Fmnn - SYS003 close error following error 00000mnn as above.

4. Examples

a) //SYS003 ACCESS MPS360,0C0='MPSWRK'
// EXEC DSINIT(MPS360)

Data set MPS360 is initialized and provided with a directory with space for

14 members. The name MPS360 is recorded in the directory descriptor, and no dummy member is set up.

```
b)//SYS003  ACCESS MPS360,OCO='MPSWRK'
  //  EXEC   DSINIT(MPS360,DIRC31,DMEMABC)
```

Data set MPS360 is initialized and provided with directory space for 44 members. The first member is set up, and is called ABC.

```
c) //SYS003  ACCESS MPS360,OCO='MPSWRK'
   //  EXEC   DSINIT(MYDATA,DIRC31,DMEMABC)
```

This is the same as example 2 except that the name in the directory descriptor is MYDATA. This data set may be used for all operations except those in which job control may be used to access a member.

```
d) //SYS003  ACCESS  MPS360,OCO='MPSWRK'
   //  EXEC   DSINIT(RENAME,MPS360)
```

Assume that MPS360 contains a directoried data set which has just been copied from a tape. The directory descriptor on the tape contains the wrong name, and job control is needed to access a member. This operation makes the names agree. At the end of the job, another RENAME run may be used to change the name back before copying the altered data set onto tape.

```
e) //SYS003  ACCESS  MPS360,OCO='MPSWRK'
   //  EXEC   COPIER(REUSE)
```

The data set is reinitialized using the name and directory size set up already in the directory. A dummy member is not created.

C. Minimal Print/Copy Program

1. General description

TPPRT is a minimal copying program primarily intended for printing tapes, but useable for any copying operation consistent with its specifications. Its advantage over COPIER is that it, with the additional save area and buffer added by the system, occupies less than one page of storage. It is thus possible to execute it under conditions when COPIER may not fit into storage.

Its input unit is SYS002, and its output unit is SYSOPT. Termination messages are written on SYSLST.

Records are read, with incorrect length indication suppressed, into a 121 character buffer. The output record length always equals the input record length, or 121 bytes, whichever is less.

SYS002 is opened with reposition and not closed. The user can thus control repositioning in multi-step jobs by means of appropriate job control cards (since subsequent opens are ignored unless job control cards have caused the tape to be closed).

Since the output is on SYSOPT, standard printer carriage control is normally in force unless overridden by an explicit access card without a LABEL card.

User options include multi-file processing, copying a fixed number of records, skipping a fixed number of records, copying until a particular record is found, and testing of the first character for valid carriage control character.

2. Use of the program

The job step consists of two cards:

```
//SYS002 ACCESS dataset
//  EXEC   TPPRT[(options)]
```

In addition an ACCESS card may be required for SYSOPT if carriage control is to be suppressed or it is to be accessed to other than the printer. No options need be supplied if none are required.

The options may be listed in any order. Their descriptions follow:

a) FILExxxx - print xxxx files. If omitted, print one file. To be used on tapes only.

b) SKIPxxxx - skip the first xxxx records. If omitted, skip no records.

c) END@zzzz - process until a record beginning with the characters zzzz is encountered. This record will not be printed. If zzzz has less than 4 characters, blanks will be assumed at the right. If this option is omitted, no test is made. This option is required when SYS002 is accessed to the system card reader, unless the COPY option is used.

d) COPYxxxx - copy xxxx records following any which were skipped. If omitted, process as per the FILE option.

e) CARR - test the first byte of each record. If it is not a valid control character, replace it by a blank. If this option is omitted, no test is made. If invalid characters appear in the first byte, action depends on the way the printer routines are designed. (If the CARR option is omitted).

Processing ends as soon as the first of the various options is satisfied.

In the FILE, SKIP, AND COPY options, if illegal characters appear among the xxxx, they are ignored. For example 99x5 is treated as 995.

3. Error conditions

Input/output errors result in termination of the job step with a message on SYSLST giving the cause of the termination (error code and unit). If the unit is stated as SYS2OPN, it refers to an error in SVC OPEN. This usually means that the SYS002 access card is missing.

4. Examples

a) //SYS002 ACCESS dsname
// EXEC TPPRT

The first file on SYS002 is printed with carriage control, but with no testing of the first bytes of the records.

b)// SYS002 ACCESS dsname
// EXEC TPPRT(FILE5,SKIP999)

The first five files on SYS002 are processed, with the first 999 records being skipped. If the five files together contain less than 1000 lines, no printing

occurs, and the tape stops at the beginning of file 6.

c) //SYS002 ACCESS SDSIPT
//SYSOPT ACCESS SDSOPT
// EXEC TPPRT(END@$$$$)

Cards are listed until a card containing $$$$ in col. 1-4 is encountered. Printer carriage control is suppressed.

d) //SYS002 ACCESS TAPE,181=
// EXEC TPPRT(SKIP9999,FILE5)
// EXEC TPPRT(FILE2)
//SYS002 ACCESS TAPE,181=
// EXEC TPPRT(COPY87,CARR)

Assuming that the first five files contain less than 10000 records, this sequence prints files 6 and 7 followed by the first 87 lines of file 1. File 1 is checked for valid carriage return codes; files 6 and 7 are not checked. Note that the second ACCESS card causes SYS002 to be closed, whereupon the OPEN in the following EXEC is executed, repositioning the tape.

D. SYSPSD Initialization Program

A special program is provided for initializing the loader-editor pseudo-directory. It is primarily used when SYSPSD is to access a queue, but may be useful at other times as well.

When executed, it first opens SYSPSD with reporition, and then writes one block set up to define a 14 member directory with name MPSPSD.

Useage is as follows:

```
//SYSPSD  ACCESS IGN
/*
//SYSPSD  ALLOC dsname,QUEUE=
//  LABEL 360
//  EXEC  PSDSET
```

These cards must appear prior to any use of SYSPSD (compiler or loader-editor).

If I/O errors occur, the job is cancelled. The following end codes may appear:

```
000000nn - I/O error code nn
00000100 - OPEN error
00000200 - CLOSE error
000002nn - CLOSE error after I/O error nn
00000300 - CLOSE error after OPEN error
```


XV. GLOBAL SYMBOLS

A - General description

This section describes the application-independent aspects of the global symbol structure.

The purpose of the global symbol structure is to provide a Physical User with a fast and convenient means of communicating information among its component Logical Users. The main components of the structure are:

1) the global symbol zone - it contains self describing beads (blocks of storage) accessible to all the users through a set of subroutines.
2) a symbol table that permits all data in the zone to be retrieved by name.
3) blocks of attributes describing the named data entities.
4) a library of general subroutines and functions that manipulate symbol table entries, attributes and data blocks.
5) application-specific libraries of routines that manipulate data and attributes in ways specific to various applications.

The global symbol zone is created by the physical user setup routine. It is a garbage zone with the same storage key as the applicable physical user. It contains symbol table blocks, attribute blocks, and data blocks. The term "garbage zone" means that provision is made for shifting around the data in the zone in order to collect all unused storage into large contiguous regions, a process called garbage collection.

All blocks are beads in the zone containing an even number of words in order to align them on double-word boundaries. A bead is a block of storage obtained from the dynamic storage allocator from within a particular zone. The first block of the symbol table is allocated when the zone is set; all other blocks are created and destroyed as needed. Additional symbol table blocks are created if needed; these are never moved during garbage collection. Their size is the same as the original block.

A user accesses a global variable through the subroutine library provided. The main argument of all the routines is a reference variable, the GSN or global symbol name. Any INTEGER*4 Fortran variable may be used as a GSN. The value of the GSN is the address of the appropriate symbol table entry and is set by several routines in the library.

Two subordinate reference variables are also defined: a name, stored as a string of 6 characters, and an integer identification number (value range 1-255). One of the primitive routines returns the values of the other two when given any one.

Each logical user header in the physical user contains the name of the GS zone (i.e. the zone pointer) and a bead pointer to the first block of the symbol table. These pointers may be obtained by the user with the EXTRACT SVC. They are normally needed only by the basic global symbol primitives (section XV-B).

The symbol table entry for a global symbol is normally the header for a list containing three elements. These elements are organized into a doubly linked list. The first element is the symbol table entry, the second is an attribute block and the third a data block. Structures involving longer lists may be created if an application requires them.

The doubly linked structure permits generalized garbage collection on the zone to be performed on all location independent blocks. Location dependent blocks may be fixed by setting their back pointer to zero. Garbage collection occurs automatically when needed.

The formats of the various blocks are shown in Figures 1, 2, and 3. The same global symbol bead format is used for both the attribute block and the data block, with the application-specific information going into the variable length user area. Word counts include both the header and the user area. Additional blocks of global symbol table (fig. 1) differ from the first block in that in the additional blocks, words 4-15 are used for table entries.

The delete flag in the sumbol table entry (fig. 2) is zero if the entry is active, and X'FF' if it has been deleted. Once a symbol has been created and then deleted, that entry may normally only be reused to create a new symbol with the same name as the old one.

Fig. 1 - Global Symbol Table Format

Word (dec)	Byte (hex)	0 / 0	1 / 8	2 / 16	3 / 24
0	00	Busy flag for this block	Address of next symbol table block		
1	04	Zero		Length of block in words	
2	08	Bead pointer to previous symbol table block		Number of active entries in this block	
3	0C	Unused			
4-15	10-3C	Work area for $GSCRT and $GSDLT			
16-n	40-nn	4-word symbol table entries			

Fig. 2 - Global Symbol Table Entry Format

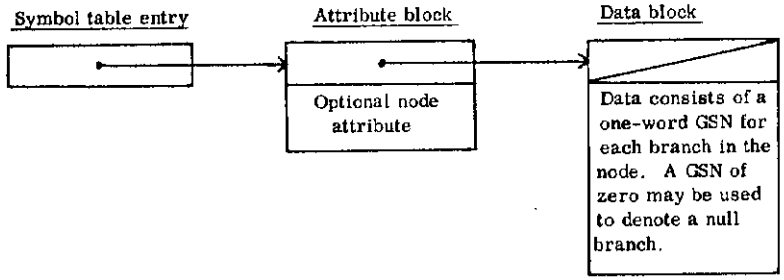
Word (dec)	Byte (hex)	0 / 0	1 / 8	2 / 16	3 / 24
0	00	Delete flag for this entry	Address of first bead for this variable		
1	04	Zero		Identification Code	
2	08	Name of symbol in EBCDIC (6 characters)			
3	0C			Identification number (1-255)	Busy flag for this entry

Fig. 3 - Global Symbol Zone Bead Format

Word (dec)	Byte (hex)	0 / 0	1 / 8	2 / 16	3 / 24
0	00	Address of next block in list (forward pointer)			
0	04	Bead pointer to previous block in list (forward pointer)		Number of words in this block	
2-xx	08-yy	user area of block			

In addition to permitting application-specific global symbolic variable definitions, the global symbol structure permits a user to define trees of variables. Each node in a tree is again a global symbolic variable. In this manner, sets of global variables may be manipulated with a single call to an action subroutine.

Figure 4 Block Structure of a Nodal Variable



Since such a structure requires that all application subroutines be recursive, certain restrictions must be placed on the use of routines that are not. Thus some routines may be used only on variables that are either terminal nodes or primitive entities, others are restricted to nodes consisting only of primitives. In addition some routines may require that a tree consist of only a certain class of entities.

The identification code in the symbol table entry determines whether a variable is a node or a primitive. In addition it supplies information about the variable or the makeup of the node.

The identification code format is as follows:

E	T	T	T	S	S	S	M	M	M	M	M	M	M	M
0	1		3	4		7	8							15

E identifies an entity as a node or a primitive.

0 - primitive 1 - node

TTT identifies the class to which a primitive entity belongs. In the case of a node it defines the class to which all the branches belong. As an example, the

values shown here define the major classes used by the nuclear physics data acquisition system,

0 - mixed node
1 - data acquisition variable
2 - display variable
3 - control variable
4-7 - undefined

SSSS further identifies the class to which a primitive or all branches belong.

0 - mixed node
1-F - specific ID codes dependent on the value of TTT

MMMMMMMM is an additional class modifier

00 - mixed node
01-FF - specific modifiers dependent on the value of TTTSSSS

The purpose of the identification code is to identify the type of a variable. In addition the homogeneity of a tree of variables can be determined from the number of non-zero modifiers in the root and subsidiary node IDs.

Information on assignment of identification code values for specific applications will be found in the literature on those applications.

Since the global symbols may be accessed effectively simultaneously by different logical users, there exists the possibility of performing conflicting operations. In order to eliminate this possibility interlocks are built in in the form of busy indicators for the entire symbol table and for each symbol table entry. Those global symbol routines which either must work undisturbed or must not interfere with other operations which may be going on test the busy indicator and wait for it to clear, if busy. When the busy indicator is found to be clear, the routine sets it before starting to work.

Under normal programming conditions, all programs must work undisturbed, and hence must first test a symbol to see that it is free and then set it busy. Assembly language programs should use the TEST AND SET instruction for this purpose.

B. Primitive Global Symbol Manipulation Routines

The following routines permit basic manipulations of the global symbol zone. They are used primarily in second-level subroutines that perform more application-specific operations. Normally users will call the second level subroutines rather than those described here. All routines are referred to by using FORTRAN CALL statements.

All arguments are full-word variables or constants aligned on word boundaries. The arguments common to all the routines are:

GSN: The value of this argument is the GSN (global symbol name) of the global variable on which the operation is to be performed. If it is an input argument, its value must be a valid GSN; if it is an output argument its initial value is ignored.

ERR: The value of this variable is set by each routine to qualify the success of the requested operation. If the requested operation was performed, the value may indicate that certain assumptions were made; if the operation was ignored the value indicates why. A list of values of this variable is given at the end of this section (table 4).

N: N is a statement label in the calling program. All the routines execute a RETURN 0 if the requested operation was performed satisfactorily. If a sufficiently severe error is encountered the routines ignore the operation and execute a RETURN 1. As usual, this argument may be omitted if not needed.

DEPTH: For most of the routines this argument specifies the depth in the list at which the operation is to be performed. The normal values of this argument are

0 - Global symbol table entry
1 - Attribute block
2 - Data block

Higher values specify further blocks in longer lists.

DEPTH must be INTEGER*4.

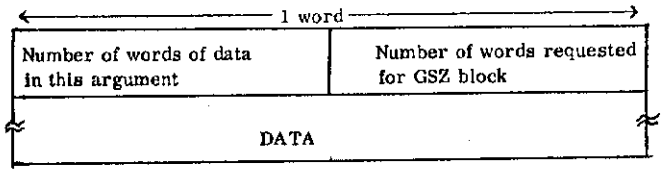
If the calling program's parameter list is too short, the primitives terminate the job step with end code 00030101.

1. Creation of new elements

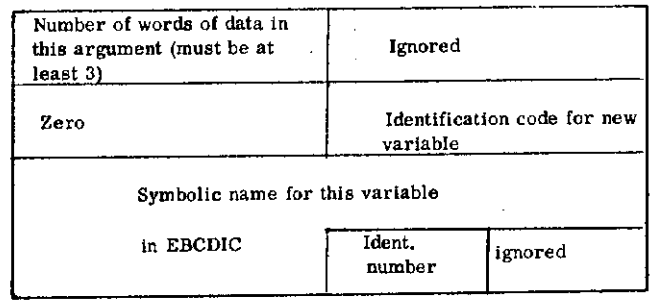
$GSCRT (GSN, ERR, &N, DEPTH, A1, A2, . . .)

This routine creates new elements in a global variable list starting at the specified depth. One block is set up for each A in the argument list. Any data in the Ai arguments is transferred to the corresponding block. Both the symbol table and the GSN are made busy during the operation.

For DEPTH >0, the Ai arguments are arrays of words formatted as follows:



If DEPTH = 0 then GSN is an output argument which will point to the symbol table entry for the newly created global variable. In this case A1 should contain the data for the sumbol table entry and is formatted as follows:



If A1 is longer than this, any remaining data are ignored. A2, A3, . . . specify the data for blocks further into the new variable list. They are of normal format (as for depth >0).

When DEPTH = 0, a new entry is made in the global symbol table, except in the following cases: a) If the symbol already exists, nothing is done and an error code is returned. b) If this name once existed and was deleted, the old entry is reused. c) If an old entry is found in which bits 0-3 of the identification code (byte 7 of the symbol table entry) are zero, it is reused regardless of the old name. This can only happen if a user program has explicitly cleared these

bits with $GSPUT.

2. Deletion of elements

$GSDLT (GSN,ERR,&N,DEPTH1,DEPTH2)

This routine deletes blocks in the variable list from DEPTH1 to DEPTH2 (DEPTH2 ≥ DEPTH1), inclusive. If DEPTH2 is out of range, then the portion of the list from DEPTH1 to the end of the list is deleted. If DEPTH1=0 the entire list is deleted, but the table entry remains intact. It is flagged with a X'FF' in its first byte. Both the symbol table and the GSN are made busy during this operation.

3. Location of the GSN, name, and ID number of a symbol

$GSFND (GSN,ERR,&N,SYMBOL,ID)

SYMBOL must be an array of 8 bytes on a fullword boundary (i.e. REAL*8 or a 2 word array of INTEGER*4, LOGICAL*4 or REAL*4). The name must be left-adjusted in this array, and padded with blanks. Only the first six bytes are significant.

If GSN≠0, the character name and identification number are returned in SYMBOL and ID, respectively. If the specified variable has been deleted, the request is processed, but the value of ERR indicates that the variable is deleted. The symbol table is made busy during this operation.

If GSN=0 and SYMBOL≠0, then GSN and identification number are returned in the appropriate arguments. Again deleted entries merely cause a flag to be set in ERR. The symbol table is made busy during this operation.

If GSN=0, SYMBOL=0, and ID≠0, the GSN and character name of the first active entry having this identification number are returned. No indication is given if other entries have this same identification number except that should a deleted entry having this number precede the active one, a flag is set in ERR and the active entry is returned. The symbol table is made busy, and in this case each GSN is also made busy while its ID number is checked. The statement SYMBOL=0 means that all six bytes of the name are binary zeroes.

4. GSN pointer manipulation

$GSEXC (GSN,ERR,&N,GSN1,DEPTH1[,DEPTH2])

This routine exchanges list pointers in GSN and GSN1 at DEPTH1 (if DEPTH2 is omitted). If DEPTH2 appears the list GSN from DEPTH1 on down is exchanged with the list GSN1 from DEPTH2 on down. Note that both DEPTH1 and DEPTH2 must take positive values. The GSN is made busy during this operation. The symbol table is not made busy.

5. Presetting of data

$GSCLR (GSN,ERR,&N,DEPTH[,VALUE])

This routine copies the full word argument VALUE into all data words of the block specified by DEPTH in the list GSN. If VALUE is omitted, it is assumed to be zero. Note that DEPTH must take a positive value. The GSN is made busy during the operation, the symbol table is not made busy.

6. Busy indicator manipulation

$GSLCK (GSN,ERR,&N,TIME)

This routine sets the specified GSN busy if it is available. If it is busy a wait of length TIME is initiated. If it becomes available before the time period elapses, it is again set busy and RETURN0 is executed. If a timeout occurs a RETURN1 is executed. The units of TIME are 1/300 second. For an infinite wait, set TIME =-1.

$GSULK (GSN,ERR,&N)

This routine sets the specified GSN not busy. It must only be used following a successful $GSLCK.

7. Data insertion and extraction

$GSGET (GSN,ERR,&N,DEPTH,VALUE,WORD[,BIT[,BITS]])

$GSPUT (GSN,ERR,&N,DEPTH,VALUE,WORD[,BIT[,BITS]])

These routines allow the caller to extract or insert bits in a global symbol list block specified by GSN and DEPTH. BIT (0-31) specifies the first bit in the specified WORD (=0-n). BITS (1-32) specifies the number of contiguous bits involved in the transfer. These bits are right-justified in VALUE. If BITS is omitted it is assumed to be 1. If BIT and BITS are both omitted, they are assumed to be 0 and 32 respectively. Note that user data begin with word 2.

These routines do not set the symbol and symbol table busy. Rather the user must call $GSLCK to set it busy first. This separation of functions is necessary because many operations require several calls to $GSGET and $GSPUT, between which the symbol must remain undisturbed. All such operations must be preceded by a call to $GSLCK, and followed by a call to $GSPUT.

Table 4 - Global Symbol Routine Error Codes in ERR

Routine	Byte	Bit	Meaning When One
$GSCRT	0-1	-	Not used
	2	0	Create operation incomplete
		1-4	Reserved
		5	GSN deleted or undefined
		6	Duplicate definition
		7	Reserved
	3	0	Not used
		1	Depth specification error
		2	Symbol could not be created
		3	Insufficient core in the GSZ
		4	GSN data missing
		5	Missing parameters in list
		6	Excess data trunacated in one or more blocks
		7	New symbol table block created
$GSDLT	0-1	-	Not used
	2	0	Delete operation incomplete
		1-4	Reserved
		5	GSN not defined
		6	GSN already deleted
		7	Reserved
	3	0	DEPTH2 less than DEPTH1
		1	DEPTH1 less than zero
		2	Block could not be deleted
		3	DEPTH1 not in range
		4	Not used
		5	Missing parameters in list
		6-7	Not used
$GSFND	0-1	-	Not used
	2	0	Find operation incomplete
		1-4	Reserved
		5	GSN not defined
		6	Not used
		7	Reserved
	3	0	Symbol not found
		1	Identification number not found
		2	Entry deleted
		3-4	Not used
		5	Missing parameters in list
		6	Parameters in error
		7	Not used

(continued on next page)

Table 4 (continued)

Routine	Byte	Bit	Meaning When One
$GSEXC	0-1	-	Not used
	2	0	Exchange operation incomplete
		1-4	Reserved
		5	GSN undefined
		6	GSN deleted
		7	Reserved
	3	0	Not used
		1	DEPTH less than zero
		2	Not used
		3	DEPTH out of range
		4	Not used
		5	Missing parameters in list
		6-7	Not used
$GSCLR	0-1	-	Not used
	2	0	Clear operation incomplete
		1-4	Reserved
		5	GSN undefined
		6	GSN deleted
		7	Reserved
	3	0	Not used
		1	DEPTH not positive
		2	Not used
		4	No data area in block
		5-7	Not used
$GSLCK and [$GSULK]	0-1	-	Not used
	2	0	Lock Unlock operation incomplete
		1-4	Reserved
		5	GSN not defined
		6	GSN deleted
		7	Reserved
	3	0-4	Not used
		5	Zero timeout specified Not used
		6	Time out error on WAIT Not used Infinite time out specified
		7	GSN already unlocked
$GSGET and [$GSPUT]	0-1	-	Not used
	2	0	Get Put operation incomplete
		1-4	Reserved
		5	GSN not defined
		6	GSN deleted and DEPTH positive
		7	Reserved

(continued on next page)

Table 4 (concluded)

Routine	Byte	Bit	Meaning When One
$GSGET and $GSPUT, continued	3	0	Not used
		1	DEPTH less than zero
		2	Not used
		3	DEPTH not in range
		4	WORD not in range
		5	Missing parameters in list
		6	BITS not in range
		7	BIT not in range

C. - Higher-Level Global Symbol Routines

Libraries of application - specific higher level global symbol routines are described in separate reports. An example of such a library is the nuclear physics data acquisition and display system.

It is desirable that any application programs make use of the primitives (or routines which call the primitives) for all global symbol manipulation. This not only provides for coding convenience, but also avoids need to consider interlocking requirements since the primitives take care of all such needs.

Page 81 is reserved for future expansion of this section.

XVI. INTER-PHYSICAL USER COMMUNICATIONS

This chapter, to be added later, will describe methods of communication between physical users.

Pages 82-86 are reserved for this chapter.

XVII. SUPERVISOR CALLS

A. Summary of Differences from 44PS

MPS supports all of the 44PS SVCs (supervisor calls) except STXIPC, STXITC, RTXIPC, and RTXITC. New SVCs have been added for dynamic storage allocation and for a new type of FETCH. Anyone using any supervisor calls must consult the design notebook for details of proper use. All SVCs destroy the contents of register 0 as well as 15.

The calling method for SVC EXTRACT has been changed. The SVC now copies a specified word or group of words from the user communication region of the logical user header into a data area in the user's program. In form, the new calling method is analogous to that for SVC INSERT (which is the same in 44PS and MPS). When the SVC is executed, register 1 must contain the address of a list of two parameters. The first is the address of the word or array of words into which the data will be copied. The second is the address of a word describing the desired section of the communication region. The first half of this word contains the number of words to be copied; the second half contains the word number of the first word to be copied. In the following example (given both in FORTRAN and assembly language), the EXEC card parameters are copied into the user's data area:

```
         LA     1,SVCPARS      INTEGER AREA(18)
         SVC    EXTRACT        INTEGER DESCR
         . . .                 INTEGER2 DESCR2(2)/18,14/
                               EQUIVALENCE(DESCR,DESCR2(1))
SVCPARS  DC     A(AREA)        . . .
         DC     A(DESCR)       CALL SVCSUB(18,IO,I15,AREA,DESCR)
AREA     DS     18F
DESCR    DS     0F             (see Section XVII-E)
DESCR2   DC     H'18'
         DC     H'14'
```

SVC GETIME is unchanged except that in MPS the time is returned in units of 1/300 second.

SVC SETIME is unchanged except that it is not useful to user programs since SVCs STXITC and RTXITC are not available. Users desiring the equivalent action should utilize a separate logical user executing the new SVC WAIT (section XVII-C). SETIME is used for internal system functions.

SVCs STXIPC and RTXIPC are replaced by equivalent facilities, described in section XVIII-B.

The calling procedures for all of the 44PS SVCs are unchanged except that SVC WAIT is generalized (see XVII-C), and the read/write SVCs have been extended to cover new types of data sets (XVII-B).

The new fetch procedure, SVC FETCH1, provides a fetch type of program load, with the relocation facilities of SVC LOAD. See the design notebook for details.

B. Input/Output SVCs

Consult the design notebook for the details of the use of the input/output SVCs. Included there is a description of the read/write level SVCs with the new types of data sets. There are also additional rules regarding device routines in EXCP programming.

A new error return code, 18(hex), is added. It means that the operation would violate memory protection.

SVC READ used with the console typewriter normally converts lower case to upper case. To suppress this conversion, set byte 1 (the second byte) of the count argument to 1 (normal value is 0).

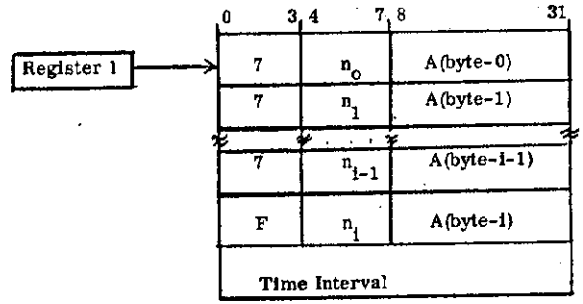
C. Generalized SVC WAIT

The WAIT supervisor call has been generalized to allow the user to wait for any event as well as completion of an I/O activity.

For an I/O wait the calling procedure is exactly the same as in 44PS

For an event wait, the calling parameter consists of a list of locations of bits in memory and a time interval. Control returns to the user program as soon as one of these bits is found to be zero, or the time interval expires, whichever occurs first.

When SVC WAIT is executed, register 1 must contain the location of a variable length list of full word addresses aligned on a full word boundary. The format of the list is as follows:



n_i is the bit-number (0-7) of the bit to be tested in the byte at the address specified in bits 8-31 of the word in the list. The time interval is specified in time-of-day units (1/300 sec.). Note that the time interval itself (and not the address of the time interval) is placed in the list. To wait for an infinite time, specify -1. If the time interval extends beyond the next overflow of the hardware clock, it is treated as -1.

To wait for an elapsed time only, it is necessary to have one dummy argument, a bit which will never change from 1 to 0, e.g.:

```
ARGS        DS      0F
            DC      X'F0'
            DC      AL3( ARGS)
TIME INT    DC      F'. . .'
```

When control returns to the caller, register 15 contains a value of 4J to indicate that bit n_J of byte -J was found to be zero. If J=i+1, the time interval expired before any of the bits changed.

Note that the SVC WAIT routine can distinguish between an I/O and a general wait by testing bit 1 of the byte pointed to by R1. Since the largest SYSUNI pointer value in MPS is 63, bit 1 is 0 for an I/O wait, and 1 for a general wait.

D. Dynamic Storage Allocator

Descriptions of the D. S. A. supervisor calls will be found in the design notebook.

User assembly language programs using the D. S. A should use the available macro instructions for all operations involving pointers and bead sizes. This enables the program to be revised to account for future changes in the D S. A. specifications by merely re-assembling with the new macro definitions.

FORTRAN programmers will usually find it to their advantage to use global symbols rather than to directly use the D. S. A. However, the D. S. A. SVCs are available through the SVCSUB routine (section XVII-D).

E. Direct Use of Supervisor Calls in FORTRAN Programs

The FORTRAN library contains a program which provides an interface between a FORTRAN program and the system's supervisor calls.

Use of this routine provides the FORTRAN programmer access to system facilities, such as the dynamic storage allocator, not normally available when programming in FORTRAN. In addition it provides an alternative means of doing input/output, other than through standard FORTRAN READ and WRITE statements.

Use of supervisor calls for I/O allows overlapping of reading, writing, and calculating, and enables greater flexibility of record size, processing of multi-file tapes, and handling of input/output errors by the FORTRAN program. On the other hand, use of SVCs instead of FORTRAN I/O leaves format conversion in the hands of the programmer.

For programs involving much complicated I/O, it is suggested that normal FORTRAN I/O be used for reading of input data and printing of results, while SVCs be used for storage and retrieval of intermediate results, taking advantage of overlapped I/O and calculating. In such a situation, a given unit should be manipulated with either the SVCs or the FORTRAN I/O statements, but not with both.

The following material describes only how to use the interface routine. The specific SVCs are described elsewhere in this manual, the design note book, and the 44PS User's Guide.

There are eight entry points: SVCSUB (for all SVCs except EXCP, WAIT, EOJS, CANCEL, UPSAND, UPSOR, and SETME), SVEXCP (for EXCP), SVWAIT (for WAIT), SVEOJS (for EOJS), SVCANC (for CANCEL), SVPSND (for UPSAND), SVPOR (for UPSOR, and SVTIME (for SETIME).

All parameters are full word variables or arrays.

a) CALL SVCSUB (number, result, err, pr1, pr2, ..., prn, list of alternate returns)

number is the SVC code number (READ=4, etc).

result is the contents of register 0 following the SVC.

err is the contents of register 15 after the SVC. If the SVC can't be handled by SVCSUB, (EXCP, WAIT, UPSOR, UPSAND, and SETIME), err is set to -4. If number is greater than 255, err is set to -8.

pr1, pr2, ... prn are the parameters specified in the description of the particular SVC, in the same order.

list of alternate returns is an optional list of alternate statement returns which will be utilized if errors occur in the SVC. RETURN 1 is executed if register 15 contains 4; RETURN 2 if it contains 8, etc. Use of these returns is optional with the user; the information is also made available in the err parameter. Note that the conditions resulting in err=-4 and -8 do not have associated alternate returns.

SVCSUB may be used to perform an EOJS or CANCEL, but it can't pass the associated termination parameter; therefore separate interfaces for these SVCs have been provided (see below).

SVCSUB does not check for non-existent SVCs (except for the check for number greater than 255). Invalid values of number will cause job cancellation by the system's SVC processor.

Note that the result and err parameters are always required, even if the SVC does not alter the corresponding registers.

Users of 44PS SVCs should consult the SVC chapter of this manual concerning changes in useage or operations under MPS.

Example: To read a record from some I/O unit:

CALL SVCSUB(4,11,ICODE,RCB,DATA,COUNT,&5,&7,&15,&23,&24,&27).

RCB, DATA, and COUNT are respectively the request control block, area into which the record is to be read, and byte count, respectively. The six alternate returns will be taken if the value of register 15 (and hence ICODE)is 4, 8, 12, 16, 20, and 24 respectively.

CALL SVCSUB(4,I1,ICODE,RCB,DATA,COUNT)

This does exactly the same job as the above example, and is included to explicitly show that the alternate returns are optional.

b) CALL SVEXCP(r15,block,alt. returns)

c) CALL SVWAIT(r15,block,alt. returns)

r15 is INTEGER*4, and is filled with the contents of register 15 following the SVC.

block is an INTEGER*4 array. For SVEXCP it contains a standard input/output block (IOB) as described in the appropriate literature; for SVWAIT it contains a list of WAIT data as described in XVII-C.

d) CALL SVEOJS(code)

e) CALL SVCANC(code)

code is any fullword variable. Its value will be placed in register 1 when the SVC (EOJS ar CANCEL) is performed, and hence will be printed by the system as the end code.

f) CALL SVPSND(value)

g) CALL SVPSOR(value)

value is a fullword whose low order byte is used as the mask in the operation. SVPSND ANDs it into the user program switch in the logical user header; SVPSOR ORs it into the switch.

h) CALL SVTIME(result, error, interval, alt. returns)

result is the value left in register 0 by the SVC (time of day of the requested interrupt).

error is the value left in register 15.

interval is the desired time interval.

alt. returns is a list of alternate statement returns corresponding to the various values in register 15.

See the description of SVC SETIME for more details.

NOTE ON LOAD-EDITING: Under normal conditions, reference to any of

the entries listed in the foregoing will cause inclusion of all of them in the user program. If spece in memory is tight, space may be saved by taking advantage of the fact that each entry is a separate independent control section. It is thus possible to include only the control section(s) needed. For example:

```
INCLUDE SVCSUB,R,(SVEOJS)
        or
INCLUDE SVCSUB,R,(SVEXCP,SVWAIT)
        or
INCLUDE SVTIME,R,(SVTIME)
```

XVIII. INTERRUPTS OTHER THAN I/O

A. External Interrupts

External interrupts are treated as simulated I/O interrupts, with each of the seven interrupts treated as a separate device on a simulated dummy I/O channel which has a channel number just above the highest real channel. The device addresses at Yale are 301-307.

The user program works with a symbolic unit accessed to the interrupt line. Any unit may be used. For example (for interrupt line 3)

```
//SYS005 ACCESS name,303=
```

To wait for an external interruption to occur, initialize an RCB (request control block) for the symbolic unit with an SVC WRITE with 2 bytes of dummy data, then do an SVC CHECK to wait for the interrupt. FORTRAN programs can call a special library program for this purpose.

To continue calculating while waiting, the program may periodically test the post request flag in the RCB. The CHECK must, however, be executed before reusing the RCB. FORTRAN programs can use SVCSVB to work in this manner.

Any interrupts which occurred before the RCB was initialized are lost.

SVCs EXCP and WAIT may be used to supply a user written device routine for an interrupt.

SVC REWIND should be executed prior to using another RCB with that unit.

This page is reserved for future expansion of this section.

B. Program Check Interrupts

Program check interrupts are handled by simulating an I/O interrupt on the program check interrupt dummy device. This device is attached to the dummy channel (3 at Yale) with the device address 3nn where nn is the logical user ID code. SYS048 is automatically accessed to this device.

Each logical user must provide its own independent means of handling its own program checks. If no procedure is provided, the job is cancelled.

In FORTRAN programs, program check interrupts are processed by the I/O routine IBCOM#.

Assembly language programs may provide their own handling in one of two ways. Both are based on the fact that the interrupt is treated as an attention interruption with the residual count in the RCB set equal to the interrupt code.

1. Do an EXCP to set up an attention IOB (input-output block) and a device routine which will get control when an interrupt occurs. Note: This device routine is subject to all the restrictions imposed on normal device routines including the fact that it may not do I/O.

2. Read/write SVCs may be used to set up a response routine which will execute in problem state and have all facilities available to it. The procedure is as follows:

An SVC WRITE to the program check interrupt device (SYS048) sets up the response routine. For this WRITE the 3 parameters are (in order): The address of the RCB, the address of an 18-word save area, and the address of a word which contains the address of the response routine.

Control returns to the calling program following the WRITE, and the RCB can be reused at that point. No CHECK is required.

When an interrupt occurs, control goes to the response routine with the interrupted program's registers and PSW already stored in the save area (the general registers first in order 0-15, followed by the PSW.) The interrupt code is in the PSW. Register 15 contains the entry point to the response program.

The response routine may use any system services. It may also alter the

registers in the save area if needed to accomplish its purpose (e.g. in replacing results of floating point overflow by a standard value). It may alter the second half of the PSW, but not the first half.

The response routine returns control to the interrupted program by executing an SVC WEOF. The registers are restored by the SVC WEOF routine.

The user program may "turn off" the program check response routine by executing an SVC REWIND to SYS048.

C. Timer Interrupts

The basic means of setting up a timer interrupt is SVC WAIT (see Chapter XVII-C).

User programs desiri ng to calculate while waiting for a timer signal should use one logical user to WAIT anda separate one to calculate. The two should be set up as one physical user, and they should communicate through a global symbol.

D. Machine Check Interrupts

The system provides a standard response to machine check interrupts described in Appendix 6 (unrecoverable system errors). There is no means to replace this with a user-written program.

E. Priority Interrupts

Priority interrupts are not supported by MPS/44.

XIX - ASSEMBLER

At present, there is no assembler in MPS. Assembly language programs must be assembled into module decks or libraries under 44PS.

A 44PS macro assembler is made available for use in writing programs for MPS and is described in separate literature.

A library of MACRO instructions is also available, and is described in detail in the design notebook. Among the general purpose macros provided are the following:

Macros for use with D.S.A. pointers and block sizes.

SYSUNIT - generate EQUs for standard unit names.

SVCALL - generate EQUs for supervisor call names.

CALL - set up FORTRAN-type call

LM - load multiple registers

STM - Store multiple registers

Macros to perform program relocation

and many others.

Appendix 1 - SUMMARY OF SIGNIFICANT DIFFERENCES IN JOB PROCESSING (COMPARED TO 44PS).

1. Since the entire logical user deck must be read onto the disk before execution can begin, the system card reader cannot be used "interactively," i.e., for entering new data as the job proceeds. All such interactive input must be through the typewriter (and later, the display and remote console).

2. The card reader is always ready to receive new logical users; they are simply queued on the disk and set into operation, multiprogrammed with current users, as soon as possible.

3. A user is disabled if all requested facilities are not available. When the facilities become available, the operator must enable the user manually.

4. Printed and punched output do not actually occur until the logical user is deleted unless the program unloads these units, as described in Chapter V-B.

Appendix 2 - INITIAL PROGRAM LOADING (IPL) PROCEDURES

A. Standard Routine IPL

1. Load the system disk onto the proper drive (check disk label).
2. Normally load the work data set disk onto the other drive.
3. Set LOAD UNIT to the system drive.
4. Ready the typewriter.
5. Hit LOAD
6. Hit REQUEST on the typewriter.
   The system types 05?*
7. Hit REQUEST, type 05 followed by EOB
8. (PROCD light is on)
   Type date, [time]
   date is any 8 characters - mo/day/yr
   time is hhmmss (hour, min. sec.)
   Time may be omitted (in which case, midnight is assumed) but the comma must be typed.
9. System types 05 > IPULSER DELETED
   INT REQ 00A**
   01 ?+
10. Logical users may be now entered through the reader, and operator commands through the console typewriter.

---

*User ID of the IPL program

** Device address of the system card reader.

+ Operator program requesting input.

B - Modifying the Device Table

Steps 1 - 7 as above

8 - type date, time, { FRESH / ALTER }

FRESH means this is a new system. Device table will now be defined. This is also used when it is necessary to run the existing system on the "wrong" drive.

ALTER - Device table will be altered.

9. System types 05?
10. Hit REQUEST, type 05 followed by EOB.
11. Enter any desired SUB and ADD commands followed by GO. These commands are exactly as in 44ps except that no blank precedes the word ADD or SUB. Step 10 must be repeated for each command.
12. Continue as in steps 9 - 10 in A).

The list of devices supported by MPS/44 may be found in the design notebook, together with the proper type designations to be used in the ADD command. All the standard 44PS devices are supported along with the 1627 plotter, 1827 data control, and 2972 data acquisition interface and control unit. The system input, printer, and punch units are the first appropriate devices encountered in the ADD commands.

To declare a device temporarily unavailable (e. g. because of equipment failure), IPL with ALTER and enter the following command in step 11:

DOWN dvadr

where dvadr is the device address. To restore such a device to use, IPL with ALTER, and enter :

UP dvadr.

Appendix 3. MESSAGES FROM SYSTEM TO USER AND OPERATOR

In general, the message procedures and formats are the same as in 44PS, and the 44PS manuals should be used for interpretation of the messages.

Messages added by MPS are described in this guide under the appropriate system components, and are in the same format as the 44PS messages.

Messages to the operator are written unit SYSLOG; messages to the user are written on SYSLST.

Certain types of unrecoverable system errors are signalled by entering WAIT state with a code in the instruction counter lights. See Appendix 6 for details.

Appendix 4. PRINTED OUTPUT FORMATS

A. General Format

The general format of the printed output is the same as in 44PS. Since printed output occurs after execution, and it is identified only by the JOB card, it is essential to provide as much information as possible on this card (JOB name and perhaps comments) to enable someone other than the owner of the job to identify the output.

B. Information Printed at Job Step Termination

The following message is printed at the end of each job step:

EOJS aa IN bb AT cccccccc dddddddd WITH eeeeeeee

aa is the error code. Values are defined below.

bb is the user status at job termination. See the description of the Operator Interaction Program LIST function (Chapter XII-B) for details.

cccccccc is the first half of the interrupted program's last PSW. except in the case of program check interrupts (in which case the last 4 digits have been altered to produce a simulated I/O interrupt and the first half of the program check old PSW is given in eeeeeeee).

dddddddd is the second half of the interrupted program's last PSW.

eeeeeeee is an end code, defined below.

The error code may have one of the following values (hex)

Code	Meaning	Equivalent 44PS Message
00	normal termination (SVC EOJS)	none
01	termination by SVC CANCEL	JA0AI
02	Invalid SVC	GA07I
03	error in SVC FETCH or FETCH1 (e.g. phase can't be found)	GA08I
04	program check	HA0nI
0C	error loading message writer	FA0CI
0D	I/O unit not operational	FA0DI
0E	sense unit check	FA0EI
0F	I/O program check	FA0FI
10	system routine can't be loaded	FA10I
FD	program check in user program check rtn	HA02I
FE	operator cancel	FB0BI
FF	operator delete	none

The meaning of the end code depends on the error code, as follows:

Error code	End Code
00	Contents of general register 1 at SVC EOJS (program's own termination code)
01	Contents of general register 1 at SVC CANCEL
02	Meaningless
03	Meaningless
04	First half of user's program check old PSW (program check interrupt code)
OC-OF	Device address
10	
FD	First half of user's program check old PSW (program check interrupt code)
FE	Meaningless
FF	Meaningless

C. Logical User Dump

A logical user dump is printed whenever the job is terminated by SVC CANCEL, operator cancel or delete, or any other abnormal means, provided that DUMP was specified on the JOB card, and when the job is terminated by the operator with a dump request, regardless of the JOB card. A sample dump is shown in figure 5 (end of this section).

Each page of the dump is headed with the Job Name, title describing the data on the page, date, termination time, and page number. (See line 1 in the illustration).

The next line (2) gives the status, error code, last PSW, and end code as described previously.

This is followed by printouts of the user communication area, accounting area, and floating point register save area of the logical user header. (Lines 3, 4, and 5 in the illustration.)

Following this is printed the most recent register save area (6). If the termination occurred within a nest of several interrupts, there will be several save areas printed (7) (the most recent first). The last of these describes the user program status. These save areas are standard system save areas, and are described in detail in section 3.6.2 of the design notebook. The main features relevant to user program diagnosis are as follows:

The first line of the save area dump gives the save area ID and the current PSW (program status word) at the time of the interrupt. The save area ID has the form aa bb, in which aa identifies the system component or logical user for which the save area was created (i.e. the user receiving control of the machine), and bb identifies the interrupted logical user. The values of aa are as follows:

00 - Job scheduler
01 - External interrupt processor
02 - SVC processor
03 - Program check handler
04 - Machine check processor
05 - Channel scheduler
06 - Not used
07 - Not used
08 through FF - Logical user

Since some conditions are processed by simulating I/O interrupts, a value

of 05 does not necessarily describe the conditions leading to the interrupt; however the error code and end code for the job termination give this information.

The PSW in the first save area is the PSW at time of termination, which also appears on the top line of the dump. If the first half of the save area ID is 05, the third and fourth bytes of the PSW contain the device address of the device causing the interrupt in the form CUU (hex). If C is the dummy channel, it is a simulated I/O interrupt. At Yale, C = 3. 301-307 are external interrupts. 308 and up are program check interuupts, with the last 2 digits being the logical user ID.

The GPR printout gives the contents of the sixteen general purpose registers at the time of the interruption.

The WORK AREA printout is the contents of the five words of working storage provided to the interrupt processor.

While the first save area is usually the most relevant to the user, this is not always the case. For example, if the interruption occurs within the SVC processor due to bad user parameters, the second save area (which contains the old PSW and registers at the time of execution of the SVC) will contain the desired information about the user's program.

Following the save areas the user's I/O unit information is printed (item 8 in the illustration). For each symbolic unit in use (a), the unit control block and file control block (b) are printed on two lines (e and f), with their bead pointer (c) and actual address (d).

Next, the areas of storage used by the user program are dumped, 1 address and 8 words per line). Each is preceded by its header (control block), the format of which is described in the design notebook. Storage is dumped in the following order:

1. Program zone primary block with the zone header (a). This is the area set up by the loader-editor.

2. Program zone extensions with their headers (12) in inverse order to that in which they were requested (most recent first).

3. Zones set up by the user program.

4. Global symbol zone for this user, if any exists.

At the conclusion of the dump, the EOJS message for the user's termination is printed, followed by the time of termination of the dump.

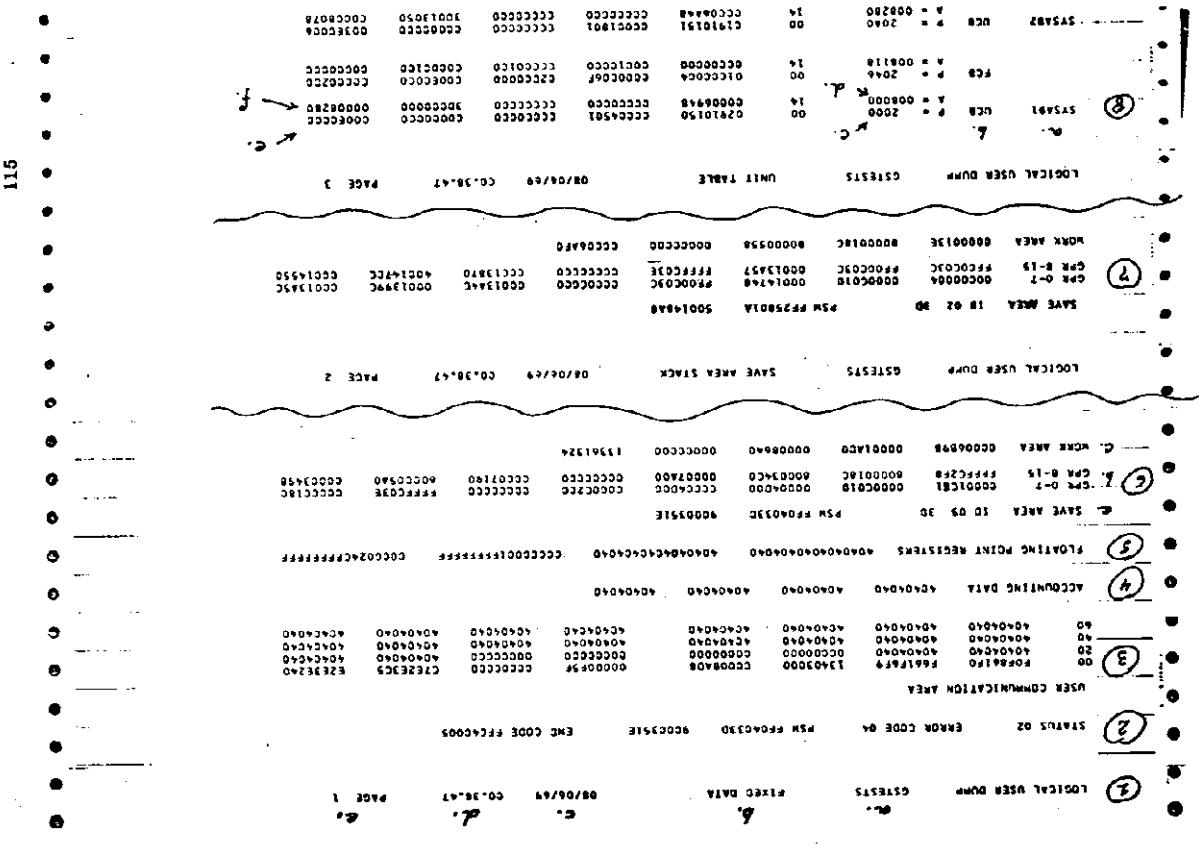


Figure 6 - Sample pages from a logical user dump (fixed data, save stack and unit table)

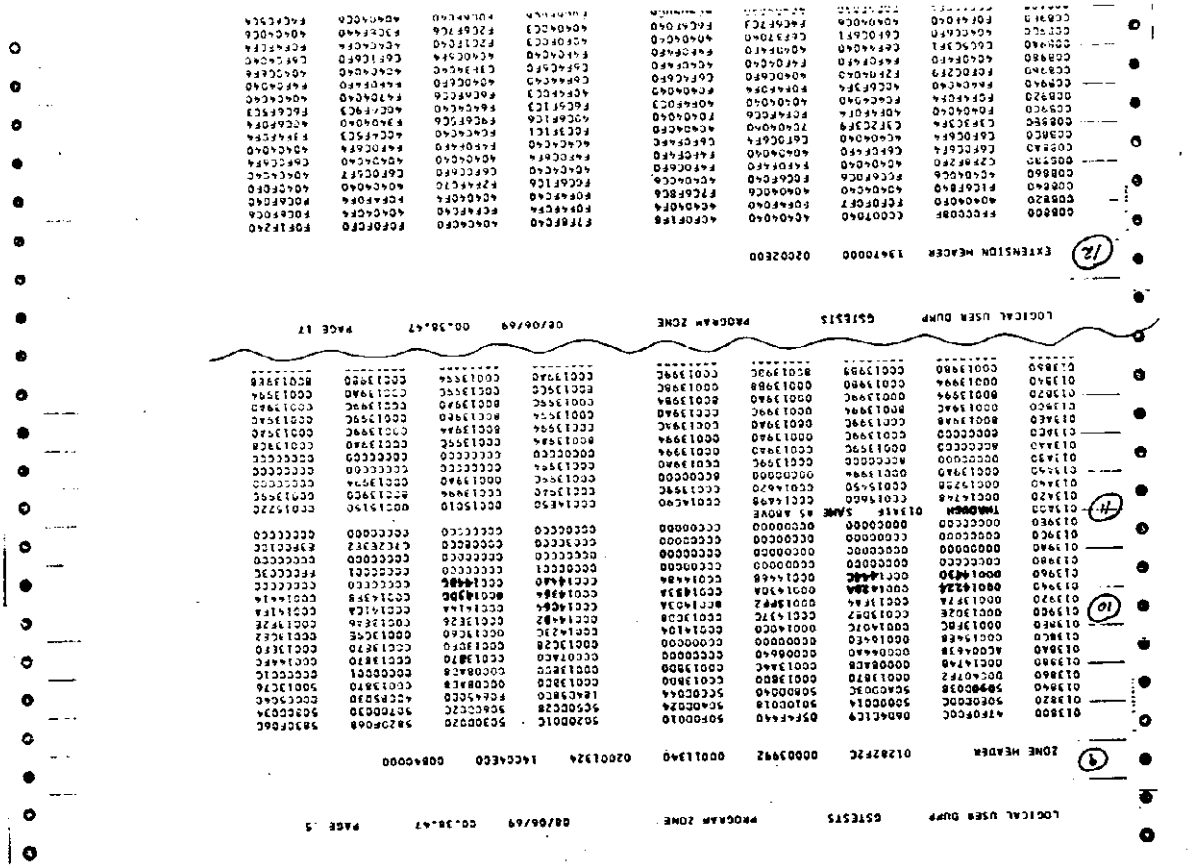


Fig. 5 Sample pages from a logical user dump (program zone and extension) (continued)

Appendix 5 - IF THE SYSTEM APPEARS TO STALL

Under some conditions, the system may appear to have come to a halt. The purpose of this appendix is to suggest means of analyzing the cause.

Under normal conditions, the system never enters wait state. When no users are active, an internal statistics program is making measurements of available time. Therefore if the WAIT light is on, an unrecoverable system error has occurred. See Appendix 6 for details. If the WAIT light is not on, it may be possible to restart if operations have stalled.

The first thing to check is the typewriter printout. One or more users may be waiting for input from the typewriter.

Next the operator program should be called to request a LIST. Analysis of the status of the various users may yield a clue. For example, if someone is waiting for a service (such as job control or loader-editor) which no one is using, it may mean that there is no core available. Deletion of one of the users should restart everyone else. If everyone is waiting for I/O, it may mean that all space for QUEUES is occupied. Attempt to initiate a printout for one of the users to relieve this condition.

The operator command C IBALL Hnn,8 may be used to check which job step user nn is executing.

If the system does not respond to the REQUEST key, it has crashed. See Appendix 6 for instructions on how to report this.

Appendix 6 - UNRECOVERABLE SYSTEM ERRORS

The system is unable to recover from certain types of abnormal conditions. Instead, it goes into wait state with an error code displayed in the instruction counter lights.

If this happens, it is important to preserve as much information as possible about the events leading to the failure. Additional information is available in the computer memory. In some cases, it may be desireable to take a memory printout (dump) using the 44PS Stand-alone Dump Program.

In the case of a machine check, the machine-check old PSW must be inspected in storage locations 30-37 (hex). The contents of memory should be preserved for study by maintenance personnel.

Following are the meanings of the codes which may appear:

Code		Meaning
1FF01	-	No more room in memory for register save areas.
1FF02	-	Machine check
1FF03	-	Invalid exit code to resident scheduler
1FF04	-	Program check in system
1FF05	-	Too many nested system waits
1FF06	-	Invalid entry in service list.

Code 1FF01 would usually mean that the system is overloaded beyond its memory capacity.

Code 1FF02 should be reported to maintenance personnel.

Codes 1FF03 through 1FF06 can result only from system software malfunctions, and should be reported to the system programming personnel with as much information as possible about the history of the system prior to the failure.

Appendix 7 - CONTROL BLOCKS AND COMMUNICATIONS REGIONS

The formats of the various system and I/O control blocks are described in the design notebook. Information omitted from there may usually be assumed to be the same as for 44PS.

The global symbol system symbol tables and other control blocks are described in chapter XV.

The user communication region of the logical user header, which may be read and manipulated with SVCs INSERT, EXTRACT, UPSOR, AND UPSAND, contains, among other things, the following information of possible interest to the normal user program: current date, pointers to global symbol zone and symbol table, elapsed CPU time for the user, job name, job step name, user program switch, inter-program and intra-program communication areas, and EXEC parameters. Details will be found in the design notebook. It should be noted that the elapsed CPU time is updated only when the user is interrupted by a higher priority user; hence in general the user may have used more time than is indicated by reading this word with EXTRACT.

Appendix 8 - DISK AND SYSTEM MAINTENANCE

All system and disk maintenance procedures which cannot be carried out with Job Control or the MPS/44 utility programs must be carried out under 44PS. The disk and data set formats follow all 44PS conventions.

In particular, disk initialization, mapping, squeezing, saving, and restoring, must be carried out under 44PS or the 44PS stand-alone programs. To aid in squeezing, all data sets allocated under MPS have expiration date 99999.

As described elsewhere the 44PS assembler with added macro facility is used for assemblies.

Maintenance requires at least a minimal 44PS hardware configuration. Two magnetic tape units are desirable if reassemblies of the system are performed frequently.

Appendix 9 - ESSENTIAL I/O EQUIPMENT

The bare minimum equipment needed to run the system is:

one single disk storgage drive

one card reader

one printer

one console 1052 typewriter

360/44 CPU with memory protect and at least 128K bytes of storage.

The system can support all devices supported under 44PS, and the more tape and disk storage available, the more users can be multiprogrammed.

In particular, the above configuration can only run self-relocating programs in the system phase library.

Execution of a program requiring the relocation facilities of the loader-editor requires at least a second disk drive on which to allocate the necessary data sets required by the loader-editor.

If the printer is down, it is possible to run jobs not requiring it by using the operator facilities in the virtual device routines to suspend printing temporarily.

POSTSCRIPT

This User's Guide describes MPS/44 as it is intended to function when the present development project is complete.

At the present time, a number of the features described are not fully implemented. Following is a list of those items, and related restrictions.

1. Physical Users

a. The physical user should only be set up after all its logical users are present and in // PAUSE state.

2. Queues and Virtual Devices

a. There is no virtual punch routine. Card output must be written on tape and punched with 44PS.

b. The virtual printer and reader should only be accessed by the standard units (SYSIPT, SYSRDR, SYSOPT, and SYSLST). General purpose queues should not be accessed by any unit which is OPENed by the program. This means that they may not be used for compiler, loader-editor, or normal FORTRAN units.

c. If a logical user terminates while a printout is in progress, the current printout is interrupted and the new one is started. When the new printout is complete, the old one resumes. To avoid this problem, terminate all logical users with a job consisting of:

```
//name JOB NODUMP
//SYSPSD ACCESS IGN
//  PAUSE  EOB TO DELETE USER
/&
```

Signal EOB only when the printer is free.

d. SVC NOTE is currently ignored if it refers to a queue.

e. Operator control of the virtual device utilities has not yet been implemented.

f. Illegal carriage control characters presently cause the virtual printer routine to loop.

3. Reentrant subroutines have not yet been implemented.

4. Job Control; Job Decks

a. An ACCESS card for SYSPSD must be present in every job deck. If SYSPSD is not actually needed, it may be accessed to IGN.

b. ACCESS cards are not accepted for SDSLOG. Programs may read and write to the console typewriter only with FORTRAN data set reference number 0 (SYSLOG).

5. FORTRAN Compiler

a. The compiler is essentially the 44PS Release 3 compiler. Any difficulties with that compiler are probably carried over into MPS.

b. Batch compilation does not work properly; each routine must be compiled in a separate job step.

c. The first card of a program should not be blank or a comment. The compiler erroneously considers this a syntax error.

d. The compiler may terminate with a program check interrupt if its tables aren't large enough. Rerun the compilation with a (larger) CORE option.

6. FORTRAN Library

a. All program check interrupts are fatal.

b. The overflow and divide check subroutines are not available yet.

7. Interrupt Handler

a. Only the EXCP method of program check response is available.

8. COPIER Utility

a. Only the first six EXEC parameters are used.

9. The Logical User Dump dumps only the user program zone.