Xerox Data Systems

XEROX

Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)

Sigma 5-8 Computers

D7

System Management Reference Manual

Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245
213 679-4511

XEROX

Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)

Sigma 5-8 Computers

System Management Reference Manual

FIRST EDITION

90 17 41A

December 1971

Price: $7.50

Printed in U.S.A.

NOTICE

This manual covers the F01 version of BPM/BTM. Much of the information contained in the manual came from the BPM/BP, RT Reference Manual (90 09 54) and the BTM/TS Reference Manual (90 15 77). Although this information is presently duplicated, the duplicate information will be deleted from those manuals during the next revision and reprint.

RELATED PUBLICATIONS

Title	Publication No.
Xerox Sigma 5 Computer/Reference Manual	90 09 59
Xerox Sigma 6 Computer/Reference Manual	90 17 13
Xerox Sigma 7 Computer/Reference Manual	90 09 50
Xerox Sigma 8 Computer/Reference Manual	90 17 49
Xerox Batch Processing Monitor (BPM)/BP, RT Reference Manual	90 09 54
Xerox Batch Processing Monitor (BPM) and Batch Time-Sharing Monitor (BTM)/ OPS Reference Manual	90 11 98
Xerox Batch Time-Sharing Monitor (BTM)/TS Reference Manual	90 15 77
Xerox Batch Time-Sharing Monitor (BTM)/TS User's Guide	90 16 79
Xerox Batch Processing Monitor (BPM)/System Technical Manual	90 15 28
Xerox BTM/BPM/UTS Overlay Loader Technical Manual	90 18 03
Xerox Symbol/LN, OPS Reference Manual	90 17 90
Xerox Meta-Symbol/LN, OPS Reference Manual	90 09 52
Xerox BASIC/LN,OPS Reference Manual	90 15 46
Xerox Extended FORTRAN IV/LN Reference Manual	90 09 56
Xerox Extended FORTRAN IV/OPS Reference Manual	90 11 43
Xerox Extended FORTRAN IV-H/LN Reference Manual	90 09 66
Xerox Extended FORTRAN IV-H/OPS Reference Manual	90 11 44
Xerox FORTRAN Debug Package (FDP)/Reference Manual	90 16 77
Xerox Extended FORTRAN/Library Technical Manual	90 15 24
Xerox Mathematical Routines/Technical Manual	90 09 06
Xerox Functional Mathematical Programming System (FMPS)/Reference Manual	90 16 09
Xerox GAMMA 3 (Matrix Generator and Report Writer for FMPS)/Reference Manual	90 17 05
Xerox CIRC-DC/Reference Manual	90 16 97
Xerox CIRC-AC/Reference Manual	90 16 98
Xerox CIRC-TR/Reference Manual	90 17 86
Xerox ANS COBOL/LN Reference Manual	90 15 00
Xerox ANS COBOL (BPM)/OPS Reference Manual	90 15 01
Xerox Sort and Merge/Reference Manual	90 11 99
Xerox Manage/Reference Manual	90 16 10
Xerox Data Management System (DMS)/Reference Manual	90 17 38
Xerox FLAG/Reference Manual	90 16 54
Xerox SL-1/Reference Manual	90 16 76
Xerox 1400 Series Simulator/Reference Manual	90 15 02
Xerox Sigma Glossary of Computer Terminology	90 09 57

Manual Content Codes: BP - batch processing, LN - language, OPS - operations, RBP - remote batch processing, RT - real-time, SM - system management, TS - time-sharing, UT - utilities.

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features may depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their XDS sales representative for details.

CONTENTS

APPENDIXES

FIGURES

TABLES

PREFACE

This manual has been designed for systems personnal at local BPM/BTM installations, including managers, analysts, and system programmers. Its purpose is to provide such personnel with a convenient reference for utilizing all of the available tools provided by the system to configure, generate, support, monitor, tune, and perhaps periodically modify their system to meet changing requirements.

It is assumed that readers are generally familiar with the contents of the BPM/BP, RT Reference Manual (90 09 54), the BTM/TS Reference Manual (90 15 77), and the BPM/BTM OPS Reference Manual (90 11 98). It is further assumed that some information given in this manual, particularly hardware requirements for a particular application, will be supplemented by XDS salesmen, XDS field analysts, and XDS field engineers.

GLOSSARY

accounting log file (:ACCTLG): a non-optional file in the :SYS account that contains a discrete record for each job processed during normal operation. Each such record contains information concerning the system resources used by the associated job (i.e., units of CPU time, I/O time, pages of output, etc.).

background area: that area of memory that is not dedicated to resident real-time foreground programs, BTM users, or the Monitor. Background memory is used for processing batch jobs.

batch job: a job that is submitted to the batch job stream through the card reader, through a remote batch terminal, or through an on-line terminal (using the BPM subsystem).

binary input: input from the device to which the BI (binary input) operational label is assigned.

checkpointed job: a real-time process wherein a partially processed background job has been saved in secondary storage along with the content of all registers and other "environment" so that the job can be restarted when the real-time task has released background memory.

conflicting reference: a reference to a symbolic name that has more than one definition.

control command: any control message other than a key-in. A control command may be input via any device to which the system command input function has been assigned (normally a card reader).

control message: any message received by the Monitor that is either a control command or a control key-in.

control programs: those Monitors, I/O handlers, diagnostics, and other utility programs that have been and will be released by XDS. Also included are standard processors such as all current assemblers and certain compilers and their associated libraries. XDS offers this type of software at no additional charge to the customer (see program products).

cooperative: a Monitor routine that transfers information between a user's program and secondary storage (also see "symbiont").

data control block (DCB): a table in the user's program that contains the information used by the Monitor in the performance of an I/O operation.

extended accounting: supplementary accounting information supplied by the installation.

external reference: a reference to a declared symbolic name that is not defined within the object module in which the reference occurs. An external reference can be satisfied only if the referenced name is defined by an external load item in another object module.

file extension: a convention that is used when certain system output DCBs are opened. Use of this convention causes the file (RAD tape, disk pack, etc.) to which the DCB is assigned to be positioned to a point just following the last record in the file. When additional output is produced through the DCB, it is added to the previous contents of the file, thereby extending the file. This is based on the assumption that the file was previously opened in the OUT mode during the current job and no new ASSIGN control commands for the DCB were encountered.

function parameter table (FPT): a table through which a user's program communicates with a Monitor function (such as an I/O function).

global symbol: a symbolic name that is defined in one program module and referenced in another.

GO file: a temporary RAD or disk pack file that contains relocatable object modules formed by a processor. Such modules may be retrieved by use of a LOAD control command in batch mode.

granule: a block of RAD or disk pack sectors large enough to contain 512 words (a page) of stored information.

job information table (JIT): a table associated with each active job. The table contains accounting information, beginning of DCB table, and temporary Monitor information.

job step: a subunit of job processing such as compilation, assembly, loading, or execution. Information from certain commands (JOB, LIMIT, and ASSIGN) and all temporary files created during a job step are carried from one job step to the next but the steps are otherwise independent.

key: a data item that uniquely identifies a record.

key-in: information entered by the operator via the operator's console.

library load module: a load module that may be combined with relocatable object modules, or other library load modules, to form a new executable load module.

linking loader: a program that is capable of linking and loading one or more relocatable object modules and/or load modules to form a nonoverlaid load module.

load map: a listing by a loader showing the location or value of all global symbols entering into the load. Also shown are symbols that are not defined or have multiple definitions.

load module (LM): an executable program formed by the overlay loader, using relocatable object modules (ROMs) and/or load modules as input information.

logical device: a peripheral device that is represented in a program by an operational label (e.g., BI or PO) rather than by specific physical device name.

monitor: a program that supervises the processing, loading, and execution of other programs.

object language: the standard binary language in which the output of a processor is expressed.

object module: the series of records containing the load information pertaining to a single program or subprogram (i.e., from the beginning to the end). Object modules serve as input to the loader.

operational label: a symbolic name used to identify a logical system device.

overlay loader: a processor that loads and links elements of overlay programs.

overlay program: a segmented program in which the element (i.e., segment) currently being executed may overlay the core storage area occupied by a previously executed element.

performance monitor: a Monitor module designed as a system management tool for monitoring time-sharing activity and for assisting the system manager in reallocating system resources to improve overall system efficiency (see "Tuning a BTM System").

physical device: a peripheral device that is referred to by a name specifying the device type, I/O channel, and device number (also see "logical device").

program products: those compilers and application programs that have been or will be released by XDS. Unlike control programs, program products are not required by all Sigma users and are therefore made available by XDS on an optional basis. Program products will be provided only to those users who execute a License Agreement for each applicable Sigma installation (see control programs).

prompt character: a character that is sent to the terminal by an on-line subsystem to indicate that the next line of input may be entered.

reentrant: an attribute of a program that allows the program to be shared by several users concurrently.

relocatable object module (ROM): a program or subprogram in XDS Sigma object language generated by a processor such as Meta-Symbol, Symbol, or FORTRAN (also see "object module").

resident program: a program that has been loaded into a dedicated area of core memory.

response time: the time between the completion of terminal input and the first program activation.

scheduler: A Monitor routine that controls on-line quantum lengths and the order in which on-line users are executed.

secondary storage: any rapid-access storage medium other than core memory (e.g., RAD or disk pack storage).

session time: the time between terminal log-in and log-out.

source language: a language used to prepare a source program suitable for processing by an assembler or compiler.

standard processors: processors that are included in a BPM/BTM system at no additional charge to the customer.

subsystem: a load module in the :SYS account to whose name is appended a colon character (:). A subsystem can be called by a time-sharing user via entry of the first two characters of the name.

symbiont: a Monitor routine that transfers information between secondary storage and a peripheral device independent of and concurrent with job processing.

symbolic input: input from the device to which the SI (symbolic input) operational label is assigned.

symbolic name: an identifier that is associated with some particular source program statement or item so that symbolic references may be made to it even though its value may be subject to redefinition.

system generation (SYSGEN): the process of creating an operating system that is tailored to the specific requirements of an installation. The major SYSGEN steps include: gathering the relevant programs, generating specific Monitor tables, loading Monitor and system processors, and writing a bootable system tape.

system library: a group of standard routines in library load module format, any of which may be incorporated in a program being formed.

system register: a register used by the Monitor to communicate information that may be of use to the user program (e.g., error codes). System registers SR1, SR2, SR3, and SR4 are current general registers 8, 9, 10, and 11, respectively.

task control block (TCB): a table of program control information built by the linking or overlay loader. The TCB is part of the load module and contains the user's temp stack and other areas for use during program execution.

tuning a system: the modification of an operating system to adjust system resources to meet changing requirements.

unsatisfied reference: a symbolic name that has been referenced but not defined.

user log file (:USERLG): an optional file in the :SYS account that contains information used to validate each user's authorization to access system facilities. It also contains accumulated statistics for each account/name combination. The file is generated by the SUPER subsystem or processor from information supplied by the system manager.

user reponse time: the time from the completion of the input command until the first character of output is produced, or the next terminal read if no output occurs. This time includes system response, queue delays due to other users, and the processing time of the users program or processor.

1. INTRODUCTION

BPM/BTM SERVICES

The Batch Processing Monitor (BPM) is an operating system that permits batch processing, remote batch processing and real-time processing. The Batch Time-Sharing (BTM) extension to the Monitor permits on-line conversational time-sharing. All processing takes place efficiently and concurrently on Sigma 5-8 computers. BPM/BTM offers

- Ability to handle up to 64 concurrent users.
- A complete recovery system coupled with preservation of user files to provide fast restart following hardware malfunction.
- For on-line users: highly efficient and extensive software, fast response time.
- For batch users: on-line entry, local entry, and high-speed remote entry.
- For installation managers: reporting facilities, system control and tuning capability, extensive error checking, and recovery features.
- For all users: comprehensive accounting and a complete set of powerful processors.

TIME-SHARING

BTM allows multiple on-line terminal users to concurrently create, debug, and execute programs using a variety of powerful and comprehensive language subsystems and facilities. The Executive routine in the Monitor provides a discrete set of services for the terminal user and provides the interface between the user and the subsystems. These subsystems and facilities include

Subsystem	Function
EDIT	Composition and modification of programs and other bodies of text.
FORTRAN	Compilationof Xerox Extended FORTRAN IV-H programs.
SYMBOL	Assembly of assembly language programs.
BASIC	Compilation and execution of programs or direct statements written in an extended BASIC language.
DELTA	Debugging of programs at the assembly language level.
LOAD	Loading of relocatable object modules (ROMs) created by on-line or off-line processors such as Symbol, Meta-Symbol, or FORTRAN.
RUN	Execution of previously formed load modules and simulation of several BPM (batch) services on-line.
BPM	Terminal batch entry.
FERRET	Listing of information about entries in the file management system and limited file manipulation.
MANAGE (TOM)	Generation of control information for batch Manage to process.

Two additional facilities allow the system manager to exercise control over the operation of the system. They are

Facility	Function
SUPER	Control of users entering the system.
BTMPM	Monitoring of system performance.

BATCH

Users with batch processing requirements have a number of choices. They may choose local batch entry at the computer site, terminal batch entry through an on-line terminal, or remote batch entry through the card reader of a remote batch terminal. In each mode of batch entry, they have access to the following batch processors:

LANGUAGE PROCESSORS

Software	Function
Extended FORTRAN IV	Compilation of Xerox extended FORTRAN IV programs.
Extended FORTRAN IV-H	Compilation of Xerox extended FORTRAN IV-H programs.
FLAG	Compilation of fast "load-and-go" FORTRAN programs.
Meta-Symbol	Assembly of high-level assembly language programs.
Symbol	Assembly of assembly language programs.

Software	Function
ANS COBOL	Compilation of programs written in English language form.
Manage	File retrieval, updating, and reporting for batch programs.
BASIC	Compilation and execution of programs or direct statements written in extended BASIC language.
SL-1	Compilation of programs written in powerful simulation language.

SERVICE PROCESSORS

Software	Function
FMGE	Creation, entry, copying, listing, deletion, or punching of files in the user's account or any account to which the M:EI or M:EO DCB is assigned.
PCL	Transfer (and conversion) of data between peripheral devices.
ROMTRAN	Elimination of forward reference items from relocatable object modules, where possible, to reduce core storage requirements for tables when loading object code.
DEFCOM	Provision of a means (primarily for foreground users) of accessing core resident data and routines in one load module by another load module.
EDCON	Creation of on-line compressed files or decompression of files. Operates as a background processor under BPM or the BTM RUN subsystem to provide background utility of functions for the text editor.
SYSGEN	Generation of BPM/BTM systems to meet user requirements.
SUPER	Control of users entering the system.
VOLINIT	Initialization of private and public disk packs.
FPURGE	Control over elimination of unwanted files.
MEDDUMP	Device-to-device data transfer to provide backup files for system recovery.
MONDUMP	Listing of crash recovery file or tape.
ELIST	Listing of error messages logged by a Monitor routine, ERRLOG.
ERRWRT	Creation of error message file.
FANALYZE	Reliability check of large file management system.

APPLICATION PROCESSORS

Software	Function
Sort-Merge	Sorting and/or merging of records in one or more files.
DMS	Organization, storage, update, and deletion of information in a centralized data base.
FMPS	Writing and execution of comprehensive linear and nonlinear programs.
GPDS	Exerimentation with and evaluation of system methods, procedures, and designs.
CIRC	Analysis of electronic circuits.
1400 Simulator	Simulation of 1400 series computers.

EXECUTION PROCESSORS

Software	Function
LOPE	Formation (in one pass) of nonoverlaid load module and, optionally, execution of that module.
LOAD	Formation (in two passes) of relocatable load module and entry of that load module in the user's element file, if a load module name is specified.

REAL-TIME

BPM provides real-time facilities concurrently with batch and/or time-sharing processing. Real-time programs may be core resident or nonresident. Nonresident real-time programs take core memory from other users to obtain needed memory. System resources such as core memory, secondary storage space, real-time clocks, I/O devices, register blocks, and trap locations may be dedicated to real-time programs.

SYSTEM MANAGEMENT FACILITIES

The manager of each BPM/BTM installation must evaluate his performance requirements before he can effectively use the system management facilities. This evaluation must take place prior to equipment selection since an effective equipment selection can be made only with complete knowledge of the intended use of the total operating system.

The performance requirements that must be defined include such things as the portions of system resources that must be devoted to batch, time-sharing, and real-time service and the type of service desired. In defining the type of service desired, the batch turnaround time that is acceptable, the interactive delays that are tolerable, and the real-time response time must be defined. Information that will affect system performance includes the number of on-line users to be allowed, the maximum core memory to be allowed each user, the maximum file space to be allowed each on-line user, and the resources to be dedicated to real-time.

Once an effective selection of equipment has been made and a BPM/BTMsystem has been installed, the system manager may exercise control over the performance of the system through several facilities. These facilities include

- System Generation
- Performance Monitoring and Control
- File Backup Control
- Log-in Supervisory Control
- Use Accounting
- Operations Control

At the time a system is generated, a number of parameters may be defined to tailor the system to the specific requirements of the installation. These parameters include

- Core size allocated to on-line users.
- Maximum number of on-line users.
- Size of terminal input/output buffers.
- Maximum file space allowed all users.
- Length of batch and on-line time quanta.
- Cutoff limits for peripheral output by batch users.
- System resources to be used by real-time programs.

After a system has been generated and put into operation, the system manager may request performance statistics supplied by the Performance Monitor. Operator key-ins allow him to change the on-line or background quanta or number of on-line users allowed to log in concurrently. Statistics supplied by the Performance Monitor

- Measure how well the system is performing.
- Warn of immediate problems (e.g., response time is becoming noticeably slower).
- Measure the importance of various parts of the system such as the relative use of various processors in terms of CPU time (this might have implications in determining whether a particular processor is dropped or whether its use justifies the effort to add new capabilities).

During operation of the system, the system manager maintains user files through the use of the FPURGE processor. This processor allows him to save, restore, purge, and list user files.

Another system management facility is the user authorization feature. This feature gives the system manager the means of adding or deleting users, and also allows him to specify the RAD and disk pack space allowed to each account/name combination, maximum batch priority, whether a user can run in batch, remote batch, real-time, or on-line, or any combination of the four.

BPM/BTM has an extensive user accounting system. Statistics maintained by account number include

- Number of batch jobs run.
- Number of terminal sessions.
- Accumulated batch job time.
- Accumulated terminal connect time.
- Accumulated CPU time.
- Accumulated overhead time.
- Accumulated I/O time.
- Number of RAD granules used.
- Number of disk pack granules used.

Accounting statistics are listed at the end of every job, and a subset of the statistics is listed when an on-line user logs off. Current values of statistics may be listed by an on-line user through the use of the FERRET subsystem. In addition, a discrete record is generated at the end of each batch job and on-line session. This record includes an accounting of resources used by the job or session.

BPM has several operational control features that allow the system manager to exercise control over operations through the computer operator. The computer operator may

- Error and abort users.
- Send messages.
- Shut down and start up on-line services.
- Control symbionts.
- Control remote batch.
- Enter and initiate real-time jobs.
- Respond to hardware errors.
- Control mounting and dismounting of magnetic tapes and private disk packs.

These functions are carried out through a console that also provides a log of overall system operation.

Thus, within reasonable limits, BPM/BTM may be modified by system management facilities to meet changing performance requirements. Beyond these limits, control must be exercised by direct management authorization and by education of users.

2. SYSTEM OVERVIEW

INTRODUCTION

The BPM/BTM operating system consists of a Monitor and a number of associated subsystems and processors (Figure 1). The Monitor provides overall supervision of program processing. The associated subsystems and processors provide specific functions such as compilation, execution, and debugging.

The Batch Processing Monitor is a partially resident supervisory program that

- Performs operator-like functions to serially process production type jobs by priority in a dedicated area of core (called background core).
- Performs services explicitly requested by the executing programs that constitute the individual jobs steps within a job.
- Prevents destruction by background programs of core dedicated to resident real-time programs (called foreground core).
- Yields control (according to priority) to real-time programs when the interrupt system so dictates, and when executing, provides unique real-time services in addition to the services common to the production type (background) program.

BPM has a file management capability that permits programs to create collections of data and/or programs (called files) which are maintained on secondary storage by the Monitor after the creating job has terminated. A file may later be accessed by any authorized background or foreground program by specifying to the Monitor the file's unique alphanumeric name (which was assigned to the file by the creating program).

The Monitor creates files on secondary storage for its own convenience as well as in response to user (program) requests. Symbiont files are collections of a not-yet-executed job's input records, or collections of an already—processed job's output records destined for unit record type output devices. Input symbiont files are produced by a small interrupt-driven Monitor program (called an input symbiont) that reads from an input device and writes a corresponding file while utilizing less than 2% of the total CPU time. The actual record reading and file writing involve I/O transfers that, in Sigma, proceed independently from and simultaneously with CPU operations. Thus, BPM may read job input from high speed secondary storage instead of from relatively slow speed devices such as card readers. BPM will also direct its output records to symbiont files instead of directly to the device. Another small interrupt-driven Monitor program (called an output symbiont) reads these output symbiont files and writes the constituent output records to the appropriate device. Once again the advantage to BPM is in the substitution of high speed secondary storage for slow speed output devices.

BPM also facilitates the construction of programs that are to be executed while only partially resident. Such programs are called "overlay programs" and at execution time consist of a "root" and one or more overlays each of which contains one of a set of mutually exclusive program elements. When an overlay program is executing, the Monitor automatically loads (from secondary storage) segments that are required but are not presently in the appropriate overlay area. The advantage to the user in creating overlay programs is the ability to execute in a smaller core area than would be required if the programs were serial. There is of course, a trade-off, in that overhead is required to load segments that results in longer execution time. In addition to permitting overlay of background and foreground programs, the Monitor is itself overlaid to reduce its size.

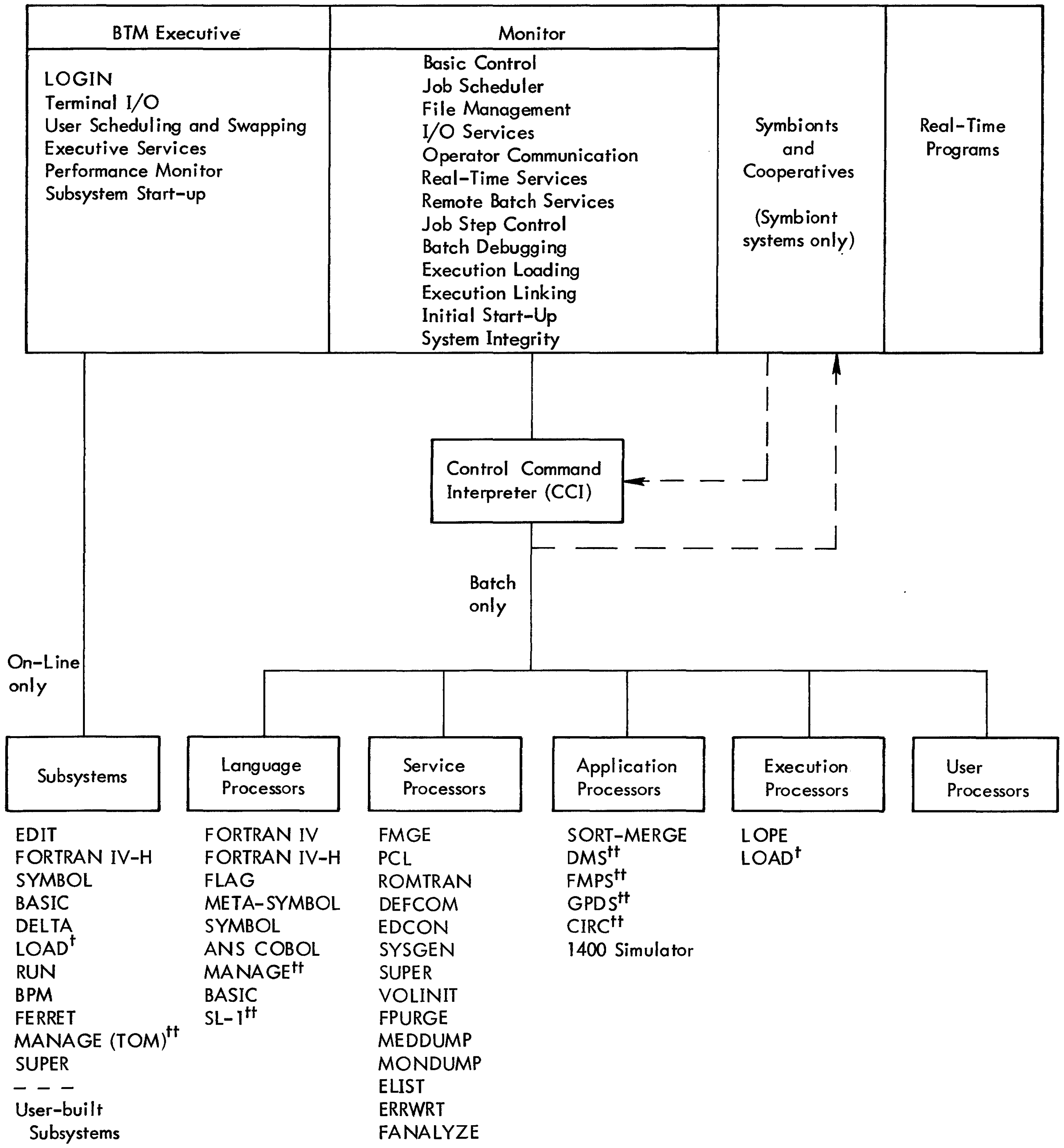
Serially processed production type jobs (sometimes called "batch" jobs) are controlled by BPM via control cards interspered throughout the stream of input cards. These control cards, which are uniquely identified to the Monitor, are commands generated by the submitter of the job to BPM to perform operator-like functions, such as a command to load (from secondary storage into background core) and execute a certain assembler program, or to dump selected areas of core memory following the execution of a program. Each job must begin with a control card identifying the job, which causes the Monitor to log and print an accounting summary for the last job and reinitialize a Job Information Table (JIT) in preparation for the new job.

Processors in BPM are programs such as assemblers (e.g., Meta-Symbol), compilers (e.g., FORTRAN), or Loaders (e.g., LOPE) that are on secondary storage at locations known to the Monitor. They may be called into background core and executed by a control card containing the processor's name (e.g., !COBOL). Additional options on these processor control cards are read by the Monitor and passed along to the processor when given control.

Subsystems in BTM are processors that may be used on-line. They are called by the BTM Executive upon receipt of the subsystem name which was entered through the terminal keyboard by the on-line user.

MONITOR SERVICES

The single most important function the Monitor performs is to provide services requested by programs operating under or in conjunction with the Monitor. I/O is usually the most requested service, and in BPM, I/O must be performed by the Monitor for background programs. I/O cannot be directly performed by background programs because of the possibility that input may be inadvertantly or maliciously directed into foreground, time-sharing, or Monitor core



† User and system libraries are available through this loader.

†† Program product.

Figure 1. BPM/BTM Operating System

areas. I/O features of the BPM system will be discussed more thoroughly below. Non-I/O services include

- Debugging assistance (i.e., conditional or unconditional snapshots of the general registers and selected core locations).
- Resource management (i.e., requesting the Monitor to provide current core limits, additional memory pages, memory protection changes for pages, simplified trap and console interrupt control, time of day and elapsed time information, operator key-ins).
- Linkages to other programs (i.e., a program may call another program in on top of itself and the called program may later restore the calling program).
- Explicit overlay segment control (i.e., the creator of an overlaid program may elect, at execution time, to explicitly request the loading of a segment whose presence is anticipated, rather than wait for the Monitor to automatically load the segment at the time it is needed).

Services are requested by programs by the execution of a CAL1 instruction. The execution of this instruction causes a trap which gives control to the Monitor. The Monitor uses the value of the register field of the CAL1 instruction and the parameter list pointed to by the instruction's address field to determine the nature of the request.

INPUT/OUTPUT SERVICES

DEVICE

The parameter list accompanying a read or write request to the Monitor for unit record type devices (including unlabeled tape) usually include, in addition to device identification, a core buffer address in the user's area into or from which I/O transfers are to occur, a byte count, a flag indicating whether the Monitor is to immediately return control to the user or is to wait until the requested I/O transfer is complete, and error or abnormal return addresses. When an immediate return is indicated, the Monitor queues (saves in a stack) the request in the event the requested device is already busy. The Monitor has the ability to queue requests for all devices and will eventually process queued requests on a priority with first-in, first-out service given to requests of the same priority. The Monitor automatically buffers transfers to unit record output devices by inserting a 34-word core buffer between the user's buffer and the destination device. When the Monitor receives a write request, it immediately transfers the data from the user's buffer to an available buffer in the Monitor's pool of output buffers. The actual I/O transfer proceeds from the Monitor buffer, thereby freeing the user's buffer for immediate alteration or update. Thus, there is no need for a user to specify "wait" on any write request to an output device. Command or data chaining may not be requested by the user and is not employed by the Monitor.

FILE

A file consists of a number of records, each of which is created by a write request. A file may have three distinct organizations depending upon how the file creator intends to later access the individual records within the file.

If each record is given a unique identifier, called a key, a record can later be accessed by requesting a read with the desired record's key specified. A file with such records has KEYED organization. If the user must access records in the order in which they appear within the file (i.e., skip or read two records in order to access the third record), the file has CONSECUTIVE organization. If the user can access a record by giving its relative physical displacement within the file (the unit of displacement is the granule which is equal to 512 words on secondary storage), the file has RANDOM organization. The user must declare his intent to create a file having one of these organization types before requesting his first write. He must also declare the name by which the file will be known. The process during which organization and file name and several other miscellaneous attributes are presented to the Monitor is known as OPEN.

The Monitor automatically introduces a 512-word blocking buffer between the user's buffer and secondary storage for files of keyed or consecutive organization. Input from or output to a random file goes through the user's buffer. User write operations are blocked, i.e., data is transferred from the user's buffer to the next available area in the blocking buffer. An actual I/O transfer occurs only when the blocking buffer is full. Similarly, when reading, an entire 512-word block is read into a Monitor blocking buffer before the user-requested record is located and transferred to the user's buffer.

Blocking and deblocking is automatic and reduces the ratio between user I/O requests and actual I/O transfers (i.e., many user requests result in only one I/O transfer) thus making more efficient use of secondary storage space. It also reduces the number of RAD or disk pack accesses, thereby eliminating I/O overhead. Any write to secondary storage consumes at least one sector regardless of how small the byte count might be. Thus, a blocked record consisting of many small user records saves space by eliminating sectors that are mostly empty. I/O requests are queued as described for device I/O.

MAGNETIC TAPE

There are two permissible types of tape operations under BPM, unlabeled and labeled. Unlabeled tape refers to tape operations that are unaltered by the Monitor. This is considered device I/O. The Monitor introduces no system information not explicitly written by the user, nor does it perform blocking/deblocking of the user's records. Labeled tape (LT) refers to tape operations that are manipulated by the Monitor in order to provide the user with advantages not present on unlabeled tape. System (Monitor) information, called sentinels, are automatically placed at the beginning and end of volumes and at the beginning and end of each file by the system. These sentinels are used by the system to verify that a user-requested volume has been mounted, used to locate a user-requested file, signify the physical end of a file, signify the physical end of volume with the continuation of the present file to the next sequential volume, or signify physical end of reel and end of tape set. Record blocking and deblocking occurs automatically just as it does for files. If two or more files exist on a LT volume set, file searching occurs over the entire set when the user requests that a specific file be opened. Volume switching is performed automatically, both during file searches and during the actual reading or writing of files split between two or more volumes. Keyed and consecutive organization is permitted, primarily to permit copying secondary storage files to LT for back-up purposes without losing any of the Monitor's control information associated with keyed and consecutive organization.

DEVICE INDEPENDENCE

Programs are often written with the intention that they be executable on any system using the same operating system (e.g., BPM standard software executes on any Sigma 5-8 configured to run BPM). However, if I/O requests within such a program were directed to a particular I/O device (i.e., a physical device address is specified) and this device was not present on all BPM systems or had a different I/O address on some systems, the programs would be constrained to execute on only those systems having the addressed devices. To eliminate this constraint, most software will address a symbolic I/O label instead of a physical device address. Special control cards (ASSIGN and STDLB) which cause symbolic I/O labels to be equated to a physical device address can be placed in front of the control card that causes the program to be loaded and executed. By changing control cards at various installations, the same program without alteration could perform I/O despite the dissimilarity of I/O configurations.

FOREGROUND OPERATIONS

A foreground program is defined as a program that enables the system to respond to and process external events in real-time; that is, within periods ranging from milliseconds to microseconds depending on the nature and urgency of the event. The foreground program is called into action by an interrupt triggered either by an external event or a clock. The interrupts have an assigned order of priority.

At the time an installation's customized Monitor system is generated (by SYSGEN), a portion of memory may be reserved for foreground programs. This foreground area differs from background memory only in memory protection details. Foreground programs may either be appended to the system at SYSGEN time and loaded into the foreground area for execution when the system is booted in, or they may be created during normal batch operations via foreground options on the !LOAD and !RUN control cards. Real-time programs may be executed in the foreground area or they may be brought into background for execution. When a real-time program is loaded in the background for execution, the background is checkpointed and the memory protection setting changed, making the background area available to the foreground program. In addition to check-pointing, BPM provides other special services for foreground programs such as the ability to connect to interrupts, take over I/O devices so that they can do their own I/O, etc.

Foreground programs are generally connected to one or more interrupts in such a way that the occurrence of an interrupt causes control to be given to the associated foreground program. In addition, a foreground program may contain a number of tasks, each individually connected to an interrupt. Each interrupt has a unique priority determined by the hardware. The priority levels and activation sequence of the interrupts control the order of execution of the foreground programs as well as the order of execution of the tasks within the program.

Foreground programs and tasks may be connected to their interrupts either directly or centrally. A directly connected foreground program or task receives control via the execution of an XPSD instruction in the associated interrupt location. A centrally connected program gets control from the Monitor after the Monitor gets control from the interrupt location. However, a centrally connected program operating in the master mode may directly connect a task within itself by storing an appropriate XPSD instruction into the appropriate interrupt location.

A more detailed consideration of the BPM and BTM and other system elements follows.

ON-LINE SUBSYSTEMS

On-line subsystems are called by entry of the first two characters of subsystem name in response to a prompt character (!) that is sent to the terminal by the BTM Executive. Upon receipt of the subsystem name (the first two characters), the BTM Executive initiates the loading of the subsystem and turns over control to the subsystem. Each of the on-line subsystems that operate under the BTM Executive is briefly described below.

EDIT

The Edit subsystem is a line-at-a-time context editor designed for on-line creation, modification, and handling of programs and other bodies of information. All Edit data is stored on RAD secondary storage in a keyed file structure of sequence numbered, variable-length records. This structure permits Edit to directly access each line or record of data.

Edit functions are controlled through single line commands supplied by the user. The command language provides for insertion, deletion, reordering, and replacement of lines or groups of lines of text. It also provides for selective printing, renumbering records, and context editing operations of matching, moving, and substituting line-by-line within a specified range of text lines. File maintenance commands are also provided to allow the user to build, copy, merge, and delete whole files. (Reference: BTM/TS Reference Manual, 90 15 77.)

EXTENDED FORTRAN IV-H

Extended FORTRAN IV-H is a one-pass high-speed compiler that is compatible with ANS standard FORTRAN and other H-level FORTRAN IV systems. In addition to the ANS standard FORTRAN features, Extended FORTRAN IV-H offers the following:

- ENTRY statement.
- Double complex data.
- FORTRAN II READ, PRINT, and PUNCH statements.
- IMPLICIT statement.
- END and ERROR options on READ statement.
- T (tab) format.
- Name LIST input/output.
- Object program listing.
- In-Line symbolic code.
- Run-time debug trace of variable assignments on BTM terminal.
- Run-time debug trace of path-of-flow on BTM terminal.
- All the debug facilities of the FORTRAN Debug Package (FDP).
- Reentrant library.
- Descriptive run-time diagnostics.
- Memory-to-memory data conversion (ENCODE/ DECODE).
- Unrestricted identifier length.
- Statement numbers as subprogram arguments.
- Adjustable formats.
- Boolean operators.

(Reference: Extended FORTRAN IV-H/LN Reference Manual, 90 09 66 and Extended FORTRAN IV-H/OPS Reference Manual, 90 11 44.)

Extended FORTRAN IV-H can call a special debugging package. The FORTRAN Debug Package (FDP) is made up of special library routines that are called by Xerox Extended FORTRAN IV-H object programs compiled in the debug mode. These routines interact with the program to detect, diagnose, and in many cases, repair program errors.

The debugger can be used in batch and on-line modes. An extensive set of debugging commands are available in both cases. In batch operation, the debugging commands are included in the source input and are used by the debugger during execution of the program. In on-line operations, the debugging commands are entered through the terminal keyboard when requested by the debugger. Such requests are made when execution starts, stops, or restarts. The debugger normally has control of such stops.

In addition to the debugging commands, the debugger has a few automatic debugging features. One of these features is the automatic comparison of standard calling and receiving sequence arguments for type compatibility. When applicable, the number of arguments in the standard calling sequence is checked for equality with the receiving sequence. These calling and receiving arguments are also tested for protection conflicts. Another automatic feature is the testing of subprogram dummy storage instructions to determine if they violate the protection of the calling argument. (Reference: FORTRAN Debug Package/Reference Manual, 90 16 77.)

SYMBOL

Symbol is a one-pass assembler that reads source language programs and converts them to object language programs. It allows forward references, literals, and external definitions. Since these items cannot be defined by a single-pass assembler, Symbol produces information that enables the loader to provide the appropriate linkages at load time. Error detection is provided and a set of mathematical subroutines is available to the assembly language user.

In addition, Symbol offers the following features:

- Self-defining constants that facilitate use of hexadecimal, decimal, octal, floating-point, and fixed-point values.
- The facility for writing large programs in segments or modules. The assembler will provide information necessary for the loader to complete the linkage between modules when they are loaded into memory at execution time.
- Values that may be specified in byte, halfword, word, and doubleword lengths.
- Instructions that are automatically aligned on word boundaries.
- The COM directive which allows the user to define instructions and table areas.

(Reference: Symbol/LN, OPS Reference Manual, 90 17 90.)

BASIC

BASIC is a compiler and programming language based on Dartmouth BASIC. It is, by design, easy to teach, learn, and use. It allows individuals with little or no programming experience to create, debug, and execute programs via an on-line terminal. Such programs are usually small to medium size applications of a computational nature.

BASIC is designed primarily for on-line program development and execution, or on-line development and batch execution. In addition, programs may be developed and executed in batch mode.

BASIC provides two user modes of operation. The editing mode is used for creating and modifying programs. The compilation/execution mode is used for running completed programs. This arrangement simplifies and speeds up the program development cycle.

Statements may be entered via a terminal and immediately executed. The principal benefit of direct execution is short simple computations. This unique capability allows an on-line terminal to be used as a "super" desk calculator.

During on-line development of programs, programs may be investigated for loop detection, snapshots of variables may be obtained, values of variables may be changed, flow of execution may be rerouted, and so on.

At compile and execute time, the user may specify if an array dimension check is to be made. In the safe mode, statements are checked to verify that they do not reference an array beyond its dimensions. In the fast mode, this time consuming check is not made. Thus, the safe mode could be used during checkout, and the fast mode could be used to speed up execution when the program reaches the production stage.

BASIC provides an image statement that uses a "picture" of the desired output format to perform editing. It also has TAB capability and a precision option to indicate the number of significant digits (6 or 16) to be printed.

An easy-to-use feature is provided to allow the user to read, write, and compare variable alphanumeric data. This is particularly important for conversational input processing.

Chaining permits one BASIC program to call upon another for compilation and execution without user intervention. Thus, programs that would exceed user core space may be segmented, and overlay techniques may be employed via the chaining facility.

In addition to the usual set of allowed matrix manipulators, Xerox BASIC provides options for input of matrixes via console or file, copying of matrixes, and the solution of simultaneous equations. Some of the matrix operations apply to vectors as well as to matrixes. (Reference: BASIC/ Reference Manual, 90 15 46.)

DELTA

Delta is designed to aid in the debugging of programs at the assembly-language or machine-language levels. It operates on core images and tables of internal and global symbols used by the programs but does not require that the tables be at hand. With or without the symbol tables, Delta recognizes computer instruction mnemonic codes and can assemble machine-language programs on an instruction-by-instruction basis. The main purpose of Delta, however, is to facilitate the activities of debugging by

- Examining, inserting, and modifying such program elements as instructions, numeric values, and coded information (i.e., data in all its representations and formats).
- Controlling execution, including the insertion of break-points into a program.
- Tracing execution by displaying information at designated points in the program.
- Searching programs and data for specific elements and subelements.

Although Delta is specifically tailored to machine language programs, it may be used to debug FORTRAN, COBOL, or any other program. Delta is designed and interfaced to BPM/BTM in such a way that it may be called in to aid debugging at any time, even after a program has been loaded and execution has begun. (Reference: BTM/TS Reference Manual, 90 15 77.)

LOAD

The Load subsystem loads XDS Sigma Standard Object Language programs consisting of relocatable object modules (ROMs) from specified element files and/or through M:BI. It may also be used to load libraries from the file :BLIB in any specified account and also the :SYS account.

The Load subsystem will load one or more object modules that have been assembled by BTM Symbol, BTM FORTRAN IV-H, standard Symbol (background), standard Meta-Symbol (background), Xerox FORTRAN IV (background), or FORTRAN IV-H (background). It will not build overlay structures and always loads modules as protection type 00 regardless of the type specified. (Reference: BTM/TS Reference Manual, 90 15 77.)

RUN

The RUN subsystem allows the on-line user to execute previously formed load modules. It simulates several BPM services that otherwise would not be available to the on-line user, allowing overlaid modules to be executed. Thus, most load modules capable of batch execution will also execute on-line. However, the execution bias must be at least one page above the lower limit of the user area. If the load module is relocatable, it will be relocated automatically.

The system load modules that execute under the RUN subsystem are

- EDCON
- LOPE
- FMGE

(Reference: BTM/TS Reference Manual, 90 15 77.)

BPM

The Terminal Batch Entry (BPM) subsystem controls the insertion of jobs into the batch job queue and is available only when BTM is operating in a symbiont environment. Input to the BPM subsystem can be from the terminal or from an existing disk file. In either case, the subsystem will allow some editing of the job file. These editing functions are intended only to provide recovery from errors of omission committed while the user is composing a small job file at his terminal. (Reference: BTM/TS Reference Manual, 90 15 77.)

FERRET

FERRET is a utility subsystem that allows the on-line user to obtain information about entries in the file management system. It also allows limited file manipulation.

Specific functions that may be performed by FERRET include

- Listing account contents.
- Testing file accessibility.
- Checking file activity.
- Listing accounting statistics.
- Sending the operator a message.
- Deleting files.
- Copying files.
- Examining files.
- Punching files on paper tape.
- Determining total granules used by files residing in user's account.
- Selectively keeping (or deleting) displayed files.

(Reference: BTM/TS Reference Manual, 90 15 77.)

TERMINAL-ORIENTED MANAGE (PROGRAM PRODUCT)

Terminal-Oriented Manage (TOM) accepts and validates request input from a terminal. The validated input is then inserted into the background job stream for processing by batch Manage.

As an adjunct to the usual batch Manage processors, TOM gives the on-line user a convenient method of accessing data bases in large business environments. However, it is directed mainly toward sophisticated Manage users who have frequent nonroutine need for reports and who want on-line validation of their retrieval and report specifications. (Reference: Manage/Reference Manual, 90 16 10.)

SUPER

SUPER provides the system manager with the ability to create, update, list, and summarize the :USERLG file that, in turn, is used by the system to control and record user activity. SUPER can be controlled from a terminal, a file, or a deck of cards.

Specific functions performed by SUPER include

- Validating user's Monitor services.
- Cancelling users.
- Summarizing statistics.
- Resetting user statistics.
- Listing authorized users.
- Listing passwords for all files in a specified account.

(Reference: BTM/TS Reference Manual, 90 15 77.)

USER-BUILT SUBSYSTEMS

BTM users may build their own on-line subsystems to supplement or replace the subsystems supplied with BTM. The rules governing the construction of these subsystems are defined in Chapter 6.

BATCH PROCESSORS

Processors that operate under BPM/BTM in batch mode (see Figure 1) are grouped as follows:

- Language Processors
- Service Processors
- Application Processors
- Execution Processors
- User-Built Processors

LANGUAGE PROCESSORS

BPM/BTM has nine language processors that operate in batch mode. These processors are outlined below.

XEROX EXTENDED FORTRAN IV

The Xerox Extended FORTRAN IV language processor consists of a comprehensive algebraic programming language, a compiler, and a large library of subroutines. The language is a superset of most available FORTRAN languages, containing many extended language features (including real-time) to facilitate program development and checkout. The compiler is designated to produce very efficient object code, thus reducing execution time and core requirements, and to generate extensive diagnostics to reduce debugging time. The library contains over 235 subprograms and is available in a reentrant version.

Xerox Extended FORTRAN IV contains all the necessary facilities to allow real-time programs to be written, debugged, and executed under BPM. These real-time facilities include: an additional language statement (CONNECT), a feature permitting reentrant object programs to be compiled, a reentrant library, and additional library routines to control interrupts, clocks, etc.

The principal features of Xerox Extended FORTRAN IV are as follows:

- Extended language features to reduce programming effort and increase range of applications.
- Extensive meaningful diagnostics to minimize debugging time.
- In-line symbolic code to reduce execution time of critical parts of the program.
- Overlay organization for minimal core memory utilization.
- Compiler produced reentrant programs.

Extended FORTRAN IV can call a special debugging program called the FORTRAN Debug Package (FDP). This is the same debugging program that is described under Extended FORTRAN IV-H. (Reference: FORTRAN Debug Package/ Reference Manual, 90 16 77.)

EXTENDED FORTRAN IV-H

A batch version of Extended FORTRAN IV-H is available to batch users. This version is essentially the same as the on-line version except that it also has real-time features. A real-time program compiled by Extended FORTRAN IV-H may be resident or nonresident, centrally connected to interrupts, and may checkpoint and use background core space. (Reference: Extended FORTRAN IV-H/LN Reference Manual, 90 09 66 and Extended FORTRAN IV-H/OPS Reference Manual, 90 11 44.)

FLAG

FLAG (FORTRAN Load and GO) is an in-core FORTRAN compiler that is compatible with the FORTRAN IV-H class of compilers. It can be used in preference to the other FORTRAN compilers when users are in the debugging phase of program development. FLAG is a one-pass compiler and uses the Extended FORTRAN IV library. Included in the basic external functions are the Boolean functions IAND (AND), IEOR (exclusive OR), and IOR (OR) which give the FORTRAN user a bit manipulation capability.

If several FLAG jobs are to be run sequentially, they may be run in a sub-job mode, thus saving processing time normally needed for the Control Command Interpreter (CCI) to interpret the associated control cards. In this mode, FLAG will successively compile and execute any number of separate programs, thereby reducing Monitor overhead.

The FLAG debug mode is a user-selected option that generates extra instructions in the compiled program to enable the user, during program execution, to detect errors in program logic that might otherwise go undetected or cause unexplainable program failure. (Reference: FLAG/Reference Manual, 90 16 54.)

META-SYMBOL

Meta-Symbol is a procedure-oriented macro assembler. It has services that are available only in sophisticated macro assemblers and a number of special features designed to permit the user to exercise dynamic control over the parametric environment of assembly. It provides users with a highly flexible language with which to make full use of the available Sigma 5-8 hardware capabilities. Meta-Symbol may be used only in batch mode.

One of the many Meta-Symbol features is a highly flexible list definition and manipulation capability. In Meta-Symbol, lists and list elements may be conveniently redefined, thus changing the value of a given element.

Another Meta-Symbol feature is the macro capability. Xerox uses the term "procedure" to emphasize the highly sophisticated and flexible nature of its macro capability. Procedures are assembly-time subroutines. Procedure definitions, references, and recursions may be nested up to 32 levels.

Meta-Symbol has an extensive set of operators to facilitate the use of logical and arithmetic expressions. These operators facilitate the parametric coding capabilities available with Meta-Symbol (parametric programming allows for dynamic specification of both "if" and "how" a given statement or set of statements is to be assembled).

Meta-Symbol users are provided with an extensive set of directives. These directives, which are commands intrinsic to the assembly, fall into three classes:

- Directives that involve manipulation of symbols.
- Directives that allow parametric programming.
- Directives that do not allow parametric programming.

A number of intrinsic functions are also included in Meta-Symbol. These give the user the ability to obtain information on both the structure and content of a assembly time construct. For example, the user can acquire information on the length of a certain list. He can inquire about a specific symbol and whether it occurs in a procedure reference. (Reference: Meta-Symbol/Reference Manual, 90 09 42 and Symbol/Reference Manual, 90 17 90.)

SYMBOL

The batch version of Symbol is essentially the same as the on-line version except for I/O conventions. It is a one-pass assembler that reads source language programs and converts them to object language programs. (Reference: Symbol/LN, OPS Reference Manual 90 17 90.)

ANS COBOL

The Xerox ANS COBOL compiler offers the user a powerful and convenient programming language facilitity for the implementation of business or commercial applications. The language specifications fully conform to the proposed ANSCII standard for the various function processing modules. Only those language elements that cause ambiguities or are seldom used have been deleted. The compiler's design takes full advantage of the unique hardware features of Sigma computers, resulting in rapid compilation of source code, rapid execution of the resulting object code, and the generation of compact programs. The result is a highly efficient programming system requiring a minimum amount of storage.

Xerox ANS COBOL contains many facilities that are either not found in other systems or, if available, are provided only at greater cost in terms of equipment required. Some of the facilities that provide more flexibility and ease of use in program development include

- Implementation of table handling mode.
- Sort/merge linkage.
- Sequential access.
- Random access linkage.
- Segmentation.
- Report writer.
- Library utilization
- Calling sequence for FORTRAN, Meta-Symbol, etc.
- Packed decimal as well as floating-point arithmetic formats.
- Data name series options for ADD, SUBTRACT, MULTIPLY, DIVIDE, and COMPUTE verbs.

The system provides the user with a comprehensive set of aids to minimize the time required to print "bug-free" programs in the form of listings. These listings include:

- The source language input to the compiler with interspersed English language diagnostic messages.
- An optional listing of the relocatable binary output, printed in line number sequence identical to the source language listing.
- A cross-reference listing, indicating by the line number where each data name or paragraph name is defined in the COBOL program and where each reference is located.

In addition, at run time, the user may use TRACE and EXHIBIT to follow execution of the procedure division. The compiler is designed to take full advantage of high-speed, random access secondary storage (e.g., RAD storage). This feature means faster job execution because of minimized I/O delays, and smaller core memory requirements because of rapid overlay service. (Reference: ANS COBOL/Reference Manual, 90 15 00.)

MANAGE (PROGRAM PRODUCT)

Batch Manage is a generalized file management system. It is designed to allow decision makers to make use of the computer to generate and update files, retrieve useful data, and generate reports without having a knowledge of programming.

Manage consists of four subprograms: Dictionary, Fileup, Retrieve, and Report. The Dictionary subprogram is a data file and is the central control element in the Manage system. It consists of definitions and control and formatting parameters that precisely describe the characteristics of a data file. The Fileup subprogram initially creates and then maintains a data file. The Retrieve subprogram extracts data from a data base file according to user-specified criteria. The Report subprogram automatically prepares printed reports from data extracted by the Manage retrieval program. (Reference: Manage/Reference Manual, 90 16 10.)

BASIC

Although BASIC is designed primarily for on-line development and execution of programs, it can be run in batch mode under BPM. All the BASIC user has to do is write his program on a coding sheet, have it punched into cards, and submit the cards along with the necessary BPM control cards to the batch job stream. (Reference: BASIC/LN, OPS Reference Manual, 90 15 46.)

SIMULATION LANGUAGE (PROGRAM PRODUCT)

The Simulation Language (SL-1) is a simplified, problem-oriented digital programming language designed specifically for digital or hybrid simulation. SL-1 is a superset of CSSL (Continuous System Simulation Language), the standard language specified by Simulation Councils, Inc., for simulation of continuous systems. It exceeds the capabilities of CSSL and other existing simulation languages by providing hybrid and real-time features, interactive debugging features, and a powerful set of conditional translation features.

SL-1 is primarily useful in solving differential equations, a fundamental procedure in the simulation of parallel, continuous systems. To perform this function SL-1 includes six integration methods and the control logic for their use. In hybrid operations, SL-1 automatically synchronizes the problem solution to real-time and provides for hybrid input and output.

Because of the versatility of Xerox Sigma computing systems and the broad applicability of digital and hybrid simulation techniques, applications for SL-1 exist across the real-time spectrum. The library concept of SL-1 allows the user to expand upon the Xerox supplied macro set and facilitates the development of macro libraries oriented to any desired application. (Reference: SL-1/Reference Manual, 90 16 76.)

SERVICE PROCESSORS

Service processors provide a variety of supporting services. Most of these services are intended for use by application programmers. Some are provided for use only by the system manager.

FMGE

The File Managment (FMGE) processor allows the batch user to enter, copy, list, delete, or punch any file in the File Directory of his account, or to copy, list, or punch any file to which the M:EI or M:EO DCB is assigned. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

PERIPHERAL CONVERSION LANGUAGE

The Peripheral Conversion Language (PCL) is a utility processor designed for operation in a batch environment under BPM. It provides for information movement among card and paper tape devices, line printers, magnetic tape devices, disk pack, and RAD storage.

PCL is controlled through command card input in the job stream. The command language provides for single or multiple file transfers with options for selecting, sequencing, formatting, and converting data records. Additional file maintenance and utility commands are provided. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

ROMTRAN

The Relocatable Object Module Translation processor reduces the amount of core storage required for tables when loading object code by stripping all possible forward reference items from user ROMs. The processor reads successive ROMs from the BI device or file and translates them until an end-of-file condition is detected. The translation is accomplished in two passes over each input ROM. The translated ROM is output to the BO device. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

DEFCOM

DEFCOM makes the DEFs and their associated values in one load module available to another load module. It accomplishes this by using a load module as input and by producing another load module that contains only the DEFs and DEF values from the root segment of the input module. The resultant load module of DEFs can then be combined with other load modules. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

EDCON

EDCON is designed to operate as a background processor under BPM or under the BTM RUN subsystem and provides background utility functions for users of the text editor (Edit). When EDCON is operating as a background processor under BPM, it may be used for

- Creating back-up files in EBCDIC or compressed format.
- Creating listings on a line printer or magnetic tape. (These listings include decimalized sequence numbers.)
- Restoring sequential back-up files for the RAD in keyed short-record format.
- Copying and merging editor-created files in the batch environment.

When EDCON is operating under the BTM RUN subsystem, it may be used for creating files in compressed format or for decompressing files. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

SYSGEN

SYSGEN is the process of creating an optimized and customized BPM system for a particular installation. The system thus generated reflects the core size, I/O configuration, processor configuration, foreground configuration, absence of optional instruction sets and substitution of instruction simulators, etc., required by the installation. The SYSGEN process, which does not necessarily have to be performed on the target machine, consists of the execution of several special SYSGEN processors which are run as batch jobs under BPM. These processors, collect, compile, load, and write the modules required for a system. The service processors are as follows:

Processor	Function
PASS1, PCL	Selects from various sources the relevant modules for system generation.
PASS2	Compiles the required dynamic tables for the resident Monitor.
LOCCT and PASS3	Store and execute load card images (by calling the loader) to produce load modules (LMs) for the Monitor and its processors.
DEF	Writes a Monitor system tape that may be booted and used.

Execution of these processors causes the creation of a customized system tape (called a PO tape). This PO tape, containing the customized Monitor and selected processors and library routines is subsequently booted on the target machine to put BPM "on the air." The various SYSGEN processors read specially formatted into cards containing details of the target configuration. This information is used by the processors to select and alter files of "skeleton" Monitors and processors which are maintained on special system tapes (called BI tapes) containing just about every element that can be included in a BPM operating system.

Bootstrap operations, including patching operations (PASS0), are performed after the required system has been generated by the other processors. (Reference: Chapters 10, 11, and 12.)

SUPER

Batch SUPER is essentially the same as on-line SUPER. It performs the same functions and accepts the same commands. The major difference is that batch SUPER is controlled from card or file input whereas on-line SUPER is controlled from terminal or file input. (Reference: Chapter 3.)

VOLINIT

The Volume Initialization processor (VOLINIT) initializes disk packs that are to be used as private volumes or public devices. Through the use of this processor, the system manager can

- Specify whether the volume being initialized is public or private.
- Specify the areas on the device that will be unconditionally flawed.
- Specify that only certain areas of the device are to be initialized.
- Specify that surface testing will be inhibited.
- Specify the account number to be inserted in the private volume's account directory.

FPURGE

The primary purpose of the File Purge processor (FPURGE) is to control and maintain user's RAD and/or disk pack files. Its most important function is to protect the integrity of user's files by periodically creating back-up files on tape, in the event that there is an irrecoverable system (software or hardware) failure. FPURGE performs the following functions:

- Saves (copies) all wanted user's files on labeled magnetic tape files.
- Restores back-up files previously saved on tape to RAD and/or disk pack.
- Logs the names of all user files by account number(s) onto the line printer.
- Purges (releases) all unwanted user's files from the RAD(s) and/or disk packs.

(Reference: BPM/BTM/OPS Reference Manual, 90 11 98.)

MEDDUMP

Like FPURGE, MEDDUMP is used primarily to provide back-up files for system recovery in the event of a system failure. The specific types of device-to-device data transfers that may be made by MEDDUMP are

- Copying private or public disk packs to disk packs (RAD-to-RAD transfers are permitted if the RADs are the same type).
- Dumping RAD or disk pack data onto disk packs or magnetic tapes in a manner that allows the device to be rebooted.
- Restoring data that was previously dumped to magnetic tape or disk packs.

(Reference: BPM/BTM/OPS Reference Manual, 90 11 98.)

MONDUMP

MONDUMP lists the contents of a crash recovery file or tape and is designed to aid the system manager in debugging system crashes. The system manager can specify both the regions to be dumped and the format of the listing. (Reference: Chapter 7.)

ELIST

A Monitor routine, ERRLOG, is called by various parts of the system that detect errors such as I/O errors, memory parity, etc. It packs error messages into a buffer for these system errors and writes the buffer to a special file on secondary storage. ELIST reads information from the packed buffers and from the file created by ERRLOG and lists the messages. (Reference: Chapter 8.)

ERRWRT

ERRWRT creates an error message file (ERRMSG). This file is searched whenever an error requiring a message is detected. The message for that error is selected from the file and listed. (Reference: Chapter 11.)

FANALYZE

FANALYZE verifies or rebuilds the granule pool bit maps, checks all linkages throughout the account and file directories and master index blocks (all levels), and produces a log on the M:LO device with information concerning the file structures. (Reference: Chapter 9.)

APPLICATION PROCESSORS

Application processors are designed to perform functions for specific applications. These applications include sorting and merging records, managing a data base, programming complex mathematical problems, simulating system methods and designs, analyzing electronic circuits, and simulating other computer hardware.

SORT/MERGE

The Sort/Merge processor provides the user with a fast, highly efficient method of sequencing a nonordered file. Sort may be called as a subroutine from within a user's program or as a batch processing job by control cards. It is designed to operate efficiently in a minimum hardware environment. Sorting can take place on from 1 to 16 keys and each individual key field may be sorted in ascending or decending sequence. The sorting technique used is that of replacement selection tournament and offers the user the flexibility of changing the blocking and logical record lengths in explicitly structured files to different values in the output file.

The principal highlights of Sort are as follows:

- Sorting capability allows either magnetic tapes, RAD, or both.
- Linkages allow execution of user own code.
- Sorting on from 1 to 16 key fields in ascending or descending sequence is allowed. Keys may be alphanumeric, binary, packed decimal, or zoned decimal data.
- Records may be fixed or variable length.
- Fixed length records may be blocked or unblocked.
- RADs may be used as file input or output devices, or as intermediate storage devices.
- Sort employs the read backward capability of the tape device to eliminate rewind time.
- User-specified character collating sequence may be used.
- Buffered input/output is used.

(Reference: Sort-Merge/Reference Manual, 90 11 99.)

DMS (PROGRAM PRODUCT)

DMS is a generalized data management system that enables the user to create an integrated data base. It is designed to be used with COBOL, FORTRAN, and Meta-Symbol processors. It simplifies programming by performing most of the I/O logic and data base management for the application programmer.

The principal features of DMS are as follows:

- The user can describe data in various data structures. Using chains, any element can be related to any other element. The data structures include lists and hierarchies (trees). The two relationships can be combined to form extensive networks of data.
- Access techniques include random, direct, indirect (relative to another record).
- Multiple secondary indexes can be defined by the user to allow records to be retrieved via any combination of secondary record keys.
- Users may construct any number of logical files or data bases within a DMS file.
- Data is described separately from the user program to facilitate management of the data base.
- Comprehensive security exists at all levels of a file.

- Journalization provides an audit trail for back-up and recovery.

- A dynamic space inventory is maintained to facilitate rapid record storage and to optimize the use of available storage space.

- Detailed data description is provided for inclusion into the user's application program to reduce programming effort.

- File I/O logic is performed for the user program, including

    1. Logical or physical record deletion.

    2. Record retrieval on random or search basis.

    3. Record insertion or modification.

(Reference: DMS/Reference Manual, 90 17 38.)

FUNCTIONAL MATHEMATICAL PROGRAMMING SYSTEM (PROGRAM PRODUCT)

The Functional Mathematical Programming System (FMPS) is a comprehensive mathematical programming package applicable to a wide variety of optimization-type problems in such areas as production scheduling, blending, process optimization, inventory control, and transportation. Four models are available: 5201, 5022, 5023, and 5024.

Model 5021 – This is the basic FMPS package and operates in a linear programming mode directed by a FORTRAN-like control language. This control language contains procedures to input, optimize (solve), output, and save the matrixes representing the constraint and objective equation that defines the problem.

Model 5022 – This model includes basic FMPS plus an optional extension. The extension provides a nonlinear programming mode plus parametric programming. The nonlinear (separable programming) mode allows problems containing nonlinear functions to be solved. Parametric programming (applicable to the lineary programming mode) allows users to observe solution variances in response to systematic changes to the objectives and restraint coefficients.

Model 5023 – This model includes basic FMPS plus an optional GAMMA 3 Matrix Generator/Report Writer program. The Matrix generator accepts problem-oriented input statements and data and constructs a problem matrix in the form required by FMPS. The report writer provides the generalized capability to prepare management reports of the solution to an FMPS problem.

Model 5024 – This model includes basic FMPS plus the extension and GAMMA 3 options.

FMPS is structured around a user-oriented control language and compiler. This FORTRAN-like language enables complete user control over all portions of the optimization process, including the many input and output options. During any FMPS operation, control is returned to the user program for corrective action should any exceptional conditions arise. The control language also permits a user to adjust parameters or tolerances, interrogate results, and alter the sequence of operations during execution. Additional procedures (input, optimization, or output) may be easily included within this control language framework.

FMPS has multiple input formats that permit a user to easily incorporate varied files and data bases into the model to be optimized. In addition to the standard input mode, FMPS will accept input prepared for MPS/360, LP/90/94, 1108LP, and CDC CDM4. FMPS will also accept binary input data produced by FORTRAN unformatted write statements. This last feature permits FMPS to be easily included in a larger user-defined optimization package.

The GAMMA 3 Matrix Generator greatly simplifies accumulation of all restraint and objective coefficients into a matrix form acceptable to FMPS. With the matrix generator, the user supplies technological data and descriptive information in tabular or list form. The list capability permits symbolic references that define the natural or logical groupings of the problem variables.

The GAMMA 3 Report Writer permits a user to describe formats and subject matter for desired solution reports. Information in the reports is obtained from user-supplied cards, text, and data entered during matrix generation, and all solution values generated by FMPS. Thus, a wide variety of reports may be produced with a minimum effort. (Reference: Functional Mathematical Programming System/Reference Manual, 90 16 09.)

GPDS (PROGRAM PRODUCT)

The General Purpose Discrete Simulator provides engineers and administrators, whose programming experience is minimal, with a system for experimenting with and evaluating system methods, processes, and designs. Providing a means for developing a broad range of simulation models, it allows organizing, modeling, and analyzing the structure of a system, observing the flow of traffic, etc. Potential applications include

- Advanced Management Planning.

- Analysis of Inventory or Financial Systems.

- Studies of Message Switching and Communications Networks.

- Risk and Capitol Investment Studies.

- Evaluation and Data Processing Systems.

- Job Shop and Queuing Studies.

Although GPDS is compatible with other simulator systems, it has a number of salient features not usually found in competitive versions. (Reference: GPDS/Reference Manual, 90 17 58.)

CIRC (PROGRAM PRODUCT)

CIRC is a set of three computer programs for electronic circuit analysis of Sigma 5-9 computers: CIRC-DC for dc circuit analysis, CIRC-AC for ac circuit analysis, and CIRC-TR for transient circuit analysis. The programs are designed for use by a circuit engineer at the installation, and require little or no knowledge of programming for execution.

CIRC can be executed under either BPM or BTM, with three modes of operation possible: conversational (on-line) mode, terminal batch entry mode, and batch processing mode. The system manager will determine which of these modes are available to the engineer, based on type of computer installation and other installation decisions. (A computer operating under BPM may only use batch mode.)

- The on-line mode offers several advantages since it provides true conversational interaction between the user and computer. Following CIRC start-up procedures, CIRC requests a control message from the user. After the control message is input (e.g., iterate a cycle of calculations with changed parameters) the computer responds (via CIRC) with detailed request for application data. These requests are sufficiently detailed to virtually eliminate misunderstandings by the engineer. This mode is highly useful in a highly interactive environment that produces a low volume of output and requires limited CPU time.

- The terminal batch entry mode allows efficient handling of high volume output and large CPU time requirements while preserving the advantages of the terminal as an input device. Two files are required: one containing all CIRC input including a circuit description and control messages, and the other directing the execution of CIRC. The job is entered from the terminal into the batch queue and treated like a batch job.

- The batch mode should generally be used for jobs involving large volumes of computations and outputs. It enables the user to concentrate on data preparation with virtually no involvement in programming considerations. The system manager can provide a set of start-up cards that never change, and these will constitute the entire interface between user and executive software. However, the batch mode offers less flexibility in experimenting with a circuit and slower turnaround time in obtaining answers.

(Reference: CIRC-AC/Reference Manual and User's Guide, 90 16 98, CIRC-DC/Reference Manual and User's Guide, 90 16 97, and CIRC-TR/Reference Manual and User's Guide, 90 17 86.)

1400 SERIES SIMULATOR

The 1400 Series Simulator provides an economical and effective solution to the program conversion problem arising because of a change in hardware. This interpretive program is designed to execute 1400 series object programs automatically as if they were run on a 1401, 1460, or 1440. Thus, an existing level of computing capability can be maintained while new processing methods that take advantage of the new, more powerful Sigma equipment are designed and implemented.

The 1400 Series Simulator simulates object code produced by SPS, FORTRAN, Autocoder, RPG, and utility routines. Almost all 1400 operations may be simulated except for I/O operations in which hardware differences make total simulation impossible. Full 1400 operator capabilities are provided. (Reference: 1400 Series Simulator/Reference Manual, 90 15 02.

EXECUTION PROCESSORS

There are two execution processors, Lope and Load. Both processors create load modules which are executable programs.

LOPE

Lope is the batch version of the load subsystem. It is a one-pass loader that forms a load module, which is an executable program, from relocatable object modules. It is not an overlay loader. If the need for an overlay loader exists, load must be used. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

LOAD

Load is a two-pass overlay loader. The first pass processes

1. All relocatable object modules (ROMs).
2. Protection types and sizes for control and dummy sections of the ROMs.
3. Expressions for definitions and references (primary, secondary, and forward references).

The second pass forms the actual core image and its relocation dictionary. (Reference: BPM/BP, RT Reference Manual, 90 09 54.)

USER PROCESSORS

BPM allows users to write special purpose batch processors to supplement or replace the batch processors supplied with the system. The rules governing the writing of these processors are described in Chapter 6.

CONTROL COMMAND INTERPRETER

The Control Command Interpreter (CCI) is brought into core whenever a control command is encountered in the background job stream or whenever a background program exits, aborts, etc. It interprets the command specified by the user and takes the necessary action. (Reference: BPM/BP, RT Reference Manual, 90 09 54 for control command functions.)

BTM EXECUTIVE

The BTM Executive is part of the Monitor root in a BPM/BTM system. It is, in turn, made up of the following:

- LOGIN
- Terminal I/O
- Scheduler and Swapper
- Executive Services
- Performance Monitor
- Subsystem Start-up

LOGIN

LOGIN admits on-line users to the system and connects the user's terminal to the Executive. If there is an authorization file, LOGIN checks to ensure that the user has on-line privileges.

TERMINAL I/O

The terminal I/O routines perform read-write buffering and external interrupt handling for I/O directed to user terminals. These routines also translate character codes, handle control characters, simulate tabs, and perform other editing tasks.

SCHEDULER AND SWAPPER

These routines decide when to swap, select the next user to be swapped in for execution, and set up the I/O command chains for swap transfers. Special algorithms control scheduling and the balance of machine use between on-line and batch.

EXECUTIVE SERVICES

The Executive Service routines provide various services for on-line terminals. These services include

- File assignments
- Subsystem calls
- Save and restore functions
- Escape and proceed functions
- Log-out functions
- Operator messages

PERFORMANCE MONITOR

The BTM Performance Monitor accumulates critical statistics during system operation. These statistics measure the performance of BTM in a given hardware configuration and provide a profile of the current user environment. The environment in any given facility will be in a fairly continuous state of change that reflects variations in the user load on a moment-by-moment basis.

The various statistical reports are received on-line and are used by the system parameter to continuously monitor the user stream. They supply the data necessary to optimize performance through system tuning. The system manager tunes the system by entering parameters through the user of operator key-ins. (Reference: Chapter 5 of this manual and the BPM/BTM/OPS Reference Manual, 90 11 98.)

SUBSYSTEM START-UP

The Subsystem Start-up routines process subsystem calls, initiate the loading of the subsystem, and turn over control to the subsystem.

MONITOR

The Monitor responds to the moment-by-moment requirements of controlling machine operation, switching between programs requiring service, and providing services at the explicit request of the user's program. The Monitor programs that perform these functions are listed below:

- Basic Control
- Job Scheduler
- File Management
- I/O Services
- Operator Communication
- Real-Time Services
- Remote Batch Service
- Job Step Control
- Batch Debugging
- Execution Loading

- Execution Linking
- Initial Start-up
- System Integrity

BASIC CONTROL

The basic control system is an I/O interrupt service and handling routine. It includes trap and interrupt handlers, routines that place requests for I/O in a queue, and basic device I/O handling routines.

JOB SCHEDULER

The job scheduler is called into memory to prepare each job to be run. It performs its functions between jobs and between job steps but is not resident while a processing program is executing. The job scheduler is initiated by an operator key-in and directed by programmer-supplied control commands. The control command input stream is under complete control of the job scheduler; it performs various functions based on the control commands processed.

There are two versions of the job scheduler. One of these reads and processes jobs sequentially. The other (for the symbiont system) can read jobs concurrently from several input devices, such as card readers, and output on several output devices, such as printers, card punch, typewriter, and paper tape. Reading and writing can occur either independently of, or concurrently with, the actual processing of the jobs.

FILE MANAGEMENT

File management controls the content and access to physical files of information. These routines perform such functions as indexing, blocking and deblocking, managing of pools of granules on RADs and disk packs, labeling, label checking and positioning of magnetic tape, and controlling access to and simultaneous use of a hierachy of files.

I/O SERVICES

I/O services consists of a set of I/O procedures that the batch user may call to perform a variety of I/O functions. These functions include

- Opening and closing a file.
- Setting error or abnormal address.
- Checking I/O completion.
- Declaring a temporary I/O file.
- Reading a data record.
- Writing a data record.
- Deleting a data record.
- Truncating a blocking buffer.
- Manipulating files (e.g., positioning a file, rewinding a tape, writing EOF, etc.).
- Specifying device setting (e.g., top of form, number of printable lines, page count, etc.).

OPERATOR COMMUNICATION

Operator communication routines provide for communication between the Monitor and the operator. They transmit messages to the operator and process key-ins received from the operator.

REAL-TIME SERVICES

Real-time services can be installed permanently as extensions of the resident Monitor or they can be dynamically loaded and initiated. The first method is used when the real-time service normally remains unchanged and is constantly operative. The dynamic approach is used when real-time operations are executed periodically or irregularly, as in an experimental laboratory.

REMOTE BATCH SERVICES

The remote batch system is an extension of the BPM symbiont system. Its purpose is to allow users at remote sites to use the central site batch facilities for the processing of their jobs. Jobs are entered at the remote site through terminals, directed through switched or nonswitched communication lines to the central site, and filed there in the input symbiont queue for later execution. In reverse manner, output is directed to the output symbiont queue, through switched or nonswitched communication lines, and reproduced at the remote site on the appropriate output device. The result at the remote site is virtually the same batch computer capability available at the central site, except that it cannot read or produce binary or compressed card decks.

JOB STEP CONTROL

Job step control routines are entered between major segments of a job. They perform the Monitor functions required between job steps such as

- Processing error exit and abort CALs.
- Handling initiated by the Monitor.
- Merging DCB assignments for execution.

BATCH DEBUGGING

Batch debugging routines provide batch programs with debugging capability through the use of procedure calls. Any batch program may take a snapshot dump of a specified segment of memory, either on an unconditional or a conditional basis.

EXECUTION LOADING

This routine (LDTRC) loads a specified load module (either one that had been partially executed and then saved as a result of the linking operation or else a new one). It then releases the core area used by the calling module and transfers control to the starting address of the called module.

EXECUTION LINKING

The Monitor LINK routine causes the calling load module's core information (i.e., program and data, except common dynamic data) to be saved on secondary storage. The calling module's core area is made available to the called module. The called module is then loaded into core (overlaying the calling program) and control is transferred to it.

INITIAL START-UP

Initialization and start-up routines are stored on tape and are booted into core storage. After they are in core, they load the Monitor root into core and turn control over to the root. The Monitor routines then complete the initialization of the Monitor and complete the patching of the system and the initialization of the swapping RAD and hardware.

SYSTEM INTEGRITY

The Monitor has a number of routines that have been included to guarantee system integrity. The objectives of these routines are, in order of importance, (1) to provide the highest possible security for user files even in the event of total system failure, (2) to provide automatic high-speed recovery in the event of a machine or software failure, and (3) to record sufficient information to isolate errors and failures caused either by hardware or software.

The major features of the BPM/BTM system integrity routines are as follows:

- Detection of malfunctions by hardware examination and software checks wherever the checks have been shown to enhance hardware error detection. Recovery from these malfunctions is through retries, operator assistance, etc.

- Logging of all malfunctions, including recovered errors and permanent failures.

- File back-up and recovery facilities to minimize the probability of losing user files and, in case of file failure, to facilitate complete recovery of the file system with a minimum of loss.

- Automatic recovery following a system failure with reasonable speed consistent with file security and the recording of information for later analysis.

- Memory dumps (MONDUMP) provide for analysis of system crashes.

ERROR DETECTION AND RECOVERY

An effective operating system must be able to detect and, whenever possible, to correct errors. It must also be capable of restarting the system if necessary. BPM uses a combination of hardware and software checks to efficiently meet these goals.

Hardware error protection features include: memory protection against accidental overwriting of Monitor and user programs, power fail-safe interrupts that ensure automatic restart in the event of power failure, memory parity checking, I/O read and write verification, and a watchdog timer to avoid instruction hangups. Detected errors are reported, logged, and if possible, recovered directly. Catastrophic failures cause an automatic system recovery if at all possible.

ERROR AND FAILURE LOGGING

Malfunction messages are maintained in a special file by system integrity routines. Messages are placed in this file whenever malfunctions are detected by the various parts of the system. Hardware malfunctions that are recorded include such things as tape errors, card reader errors, and memory parity errors. Software malfunctions that are recorded include the failure of software checks on RAD addresses contained in index blocks and improper linkage of linked file blocks.

ERROR LOG ANALYSIS

The error log analysis program is called into execution by the system manager on a periodic basis. The function of the analysis program consists of retrieving and listing the contents of the error file. These summaries may be used by the Customer Engineer to aid in preventive maintenance of the system.

AUTOMATIC RECOVERY AFTER SYSTEM FAILURE

The system recovery function is provided to restore BPM/BTM to operational status very quickly following a system failure. Recovery consists of cleaning up all open-ended information (both user and system oriented information) and restarting the system at its initialization.

Whenever various hardware and software errors are detected, the recovery routine is automatically entered. Manual entry is also provided for use when the system cannot automatically recover.

When the recovery routine is entered, no functions of the normal operating system are assumed to be operating. Some routines of the normal system are duplicated in the recovery routine but, for automatic recovery, a small resident recovery driver is required intact. This driver brings in the bulk of the recovery routine, overlaying the pure procedure portion of the system. Certain Monitor tables are also required intact. This is verified where possible. If there is any doubt as to the veracity of critical tables, the operator is informed that the recovery attempt has been cancelled and is asked if he wishes to attempt RECOVERY 2. If so, or if the operator bootstraps the system from the system device when the system is not quiescent, RECOVERY 2 is initiated. For a BPM system, a request for magnetic tape is issued and core is dumped on tape. For a BTM system, core is dumped to the swapping device. If neither RECOVERY nor RECOVERY 2 is successful, the operator is notified to reboot and restore the system with the most recent back-up volumes.

The recovery routine then performs the following functions:

- Displays cause of failure.
- Takes a full core dump for later analysis.
- Closes all open files with default options.
- Saves all output partial symbiont files and releases partial input.
- Saves error log.
- Informs users of interruption.
- Saves time, data, error log pointers, accounting information, symbiont file directory, disk granule usage map, and executive communication.
- Restarts system and restores items saved above.

When functions cannot be performed, they are noted on the operator's console. If the function is considered minor, recovery continues. If it is connected with file operations, the file identification is noted and recovery proceeds.

CRASH ANALYSIS

In the event of a "hot" system restart, one of the functions of the recovery procedure is to dump the contents of core memory onto RAD or magnetic tape storage. This information is available for later analysis by system programmers and by a special program designed to print in labeled form the contents of the Monitor's control tables.

The crash analysis program is a user program that is called by the system manager to analyze the last core image filed on disk. This program is written in such a way that additional tests may be included as they are found to be useful. Initially, it provides the following service and tests:

- Prints PSD and register contents at point of error.
- Prints direct cause of error.
- Runs some of the same checks that are used to test the dynamic integrity of the system.
- Prints the contents of the critical Monitor tables.
- Prints the contents of the current JIT.
- Prints a hexadecimal dump of core memory. Instruction mnemonics, EBCDIC characters, and global symbols can optionally be included as part of the dump of the Monitor root (see Chapter 7).

SYMBIONTS AND COOPERATIVES

A BPM system need not be a symbiont system. However, if much input and output is to be processed through slow-speed unit record-type peripherals a symbiont system may be desirable. A symbiont system provides buffering of unit record I/O on secondary storage. Symbiont routines transfer data from the card reader or paper tape reader to secondary storage and from secondary storage to the card punch, line printer, or paper tape punch. Input cooperatives intercept card or paper tape read commands in user programs and transfer data from secondary storage where it was stored by the symbiont routines. Output cooperative routines intercept output directed from a user program to a line printer, card punch, or paper tape punch and transfers the data to secondary storage.

A symbiont system must have a set of symbiont and cooperative routines.

REAL-TIME PROGRAMS

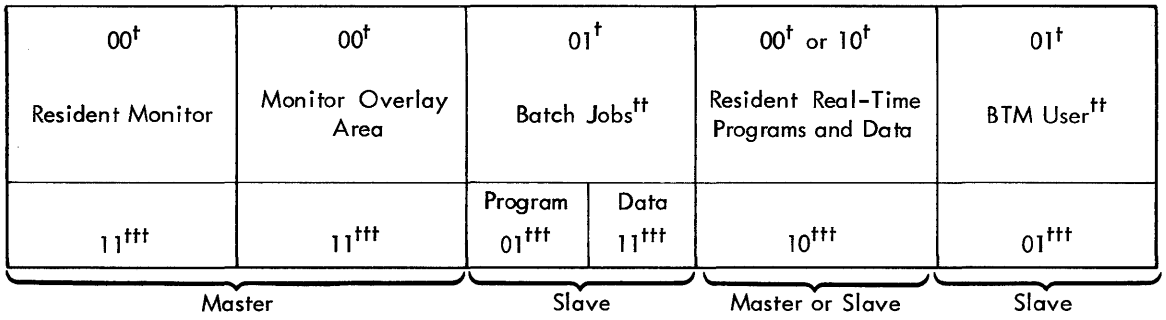
BPM and BTM users may run real-time programs concurrently with batch and time-shared programs. At installation time, a real-time process is assigned machine facilities on a dedicated basis. These facilities include secondary storage and core memory residency, external interrupt lines, and CAL trap locations. Such allocation remains in force until either the process or the computer operator terminates the program.

MEMORY LAYOUT

A typical layout of physical memory is shown in Figure 2. Although this is similar to the actual layout, it should not be assumed to be exact.

BPM/BTM makes full use of Sigma write locks to protect certain core areas from inadvertent stores. As is indicated in Figure 2, the Monitor operates with a key of 00 and can store anywhere. Batch jobs operate with a key of 01 and can store only in the batch data area. Resident real-time programs operate with either a key of 00 or a key of 10 and can store only in the real-time area. BTM users operate with a key of 01 and can store into the BTM area.

BPM/BTM also makes full use of the master/slave modes of Sigma hardware. The master mode is the basic operating mode of the computer. When a program is operating in this mode, all instructions can be executed. The slave mode is the problem-solving mode of the computer. When a program is operating in this mode, certain privileged instructions cannot be executed. These privileged instructions control the basic operating conditions of a Sigma computer. Figure 2 shows which parts of the system operate in the master mode (Monitor and possibly real-time) and which parts operate in the slave mode (batch, time-sharing, and possibly real-time).



00[t] Resident Monitor	00[t] Monitor Overlay Area	01[t] Batch Jobs[tt]		00[t] or 10[t] Resident Real-Time Programs and Data	01[t] BTM User[tt]
11[ttt]	11[ttt]	Program 01[ttt]	Data 11[ttt]	10[ttt]	01[ttt]
Master		Slave		Master or Slave	Slave

[t] Keys (unused key: 11)

[tt] Batch and BTM locks are set to 01 only when programs are being executed in the partitions. Whichever partition is not executing has all locks set to 11.

[ttt] Locks (unused locks: 00)

Figure 2. Typical Memory Layout (not to scale)

3. SUPERVISOR PROCESSOR (SUPER)

INTRODUCTION

SUPER provides the ability to create, update, list, and summarize the :USERLG file that, in turn, is used by the system to control and record user activity (see "Account Authorization" in Chapter 4). The :USERLG file contains information about each user's authorization for accessing system facilities and accumulated statistics for each account. If the :USERLG file has been created, it is located in the :SYS account and may only be accessed by :SYS. The file has no password. If the :USERLG file is not present in the system, any batch job can run and/or any on-line user can log in.

When a batch !JOB command is encountered or an on-line user responds to a LOGIN request, the :USERLG file is searched for the specified account number and name. If a batch user is not authorized or the priority specified on the !JOB command exceeds the maximum authorized for that user, the user will be aborted. If an on-line user is not authorized, the user will be asked to log in again. SUPER is available as either a batch processor or as an on-line subsystem. In batch mode, SUPER can be controlled from card input or a file; in on-line mode, SUPER can be controlled from the Teletype® or a file.

BATCH OPERATION

There are two modes of operation when running under batch: the control mode and function mode. If SUPER is being run from a file, the M:SI DCB must be assigned to the file prior to entering the control mode. SUPER begins execution in the control mode and requires a control record that defines one of six possible function options that the program can perform. The six options are

Name	Function
USERS	Authorize users
KILL	Cancel user authorization
STATS	Output user statistics summary
DELSTATS	Delete (reset) user statistics
LIST	List authorized users
PASSWORD	List passwords of user files

When SUPER has read a control record, it enters the corresponding function mode and reads specification cards that define the various services the function is to perform.

To exit from a function back to the control mode, an !EOD record from either cards or a file is required.

To exit from the control mode back to the Monitor (end the run), a record either with an X in byte 0 and a blank in byte 1 or with an EOF is required.

ON-LINE OPERATION

The Executive command

```
! SUPER
```

will cause SUPER to output

```
! FILE ID:

:
```

If SUPER is to be run from a file, the user must then type in the file ID in the form

```
file name[(account[,password])]
```

If SUPER is to be run from a Teletype, a RET causes SUPER to type out

```
ENTER OPTION:

:
```

The user then types in either one of the six function options described below or an X to exit to the Executive. Typing in one of the options causes SUPER to prompt with a > character. The user responds by typing in the proper specification record. To return to the control mode from the function mode, the user types in either a RET or LF character.

SUPER FUNCTIONS

USERS This function validates users for Monitor services specified on a control record. When the USER mode is entered, the defined users will be validated only for those services actually specified. The record used to enter the USER mode has the format.

```
: U[SERS][,BCH][,BTM][,FGD][,RBT]
```

where

BCH specifies batch jobs are to be validated for defined users.

BTM specifies time-sharing jobs are to be validated for defined users.

® Registered trademark of the Teletype Corporation.

FGD specifies real-time jobs are to be validated for defined users.

RBT specifies remote batch jobs are to be validated for defined users.

Any or all of services may be validated at one time. Note that the "U" in :U[SERS] is the first character in the record.

A specification record must follow for each user to be validated for the services specified by USERS. Each specification record has the format

```
>account,name[[(extacc)],[password],[batch pri],
   [RAD granules],[disk granules]]
```

where

account has a maximum of 8 characters.

name has a maximum of 12 characters.

extacc has a maximum of 24 characters (excluding parentheses).

password has a maximum of 8 characters.

batch pri is the batch priority, 0 through F.

RAD and disk pack granules may specify a maximum of 65,535 characters.

The only mandatory fields are "name" and "account". Null fields (those containing no characters) cause one of two possible actions:

1. If the record for a given user did not previously exist, the null fields are set to zeros or blanks.

2. If the record existed previously, the null fields are left as they were.

In any case, the authorized services are reset to those in effect for the current run.

The example

```
: USERS,BCH

>A,B(xxx), PASS,A,100,100

>!EOD

: USERS,BTM

>A,B,,F,,0

>!EOD
```

would cause a user to be validated for time-sharing (only) as if the following run was made:

```
: USERS,BTM

>A,B,(xxx),PASS,F,100,0

>!EOD
```

To exit from the USER mode, either a zero-length record (a (RET) character when using SUPER on-line) must be read or an !EOD must be input.

KILL This function cancels an authorized user from Monitor services. The format of the KILL record is

```
:K[ILL]

>account,name
   .
   .
   .
```

All account,name specifications that follow are deleted from the :USERLG file and are thereby denied access to any Monitor services.

To exit the KILL mode, either a zero-length record or an !EOD must be input.

STATS This function causes all statistics to be summarized. The format of the STATS record is

```
:S[TATS]

>option
   .
   .
   .
```

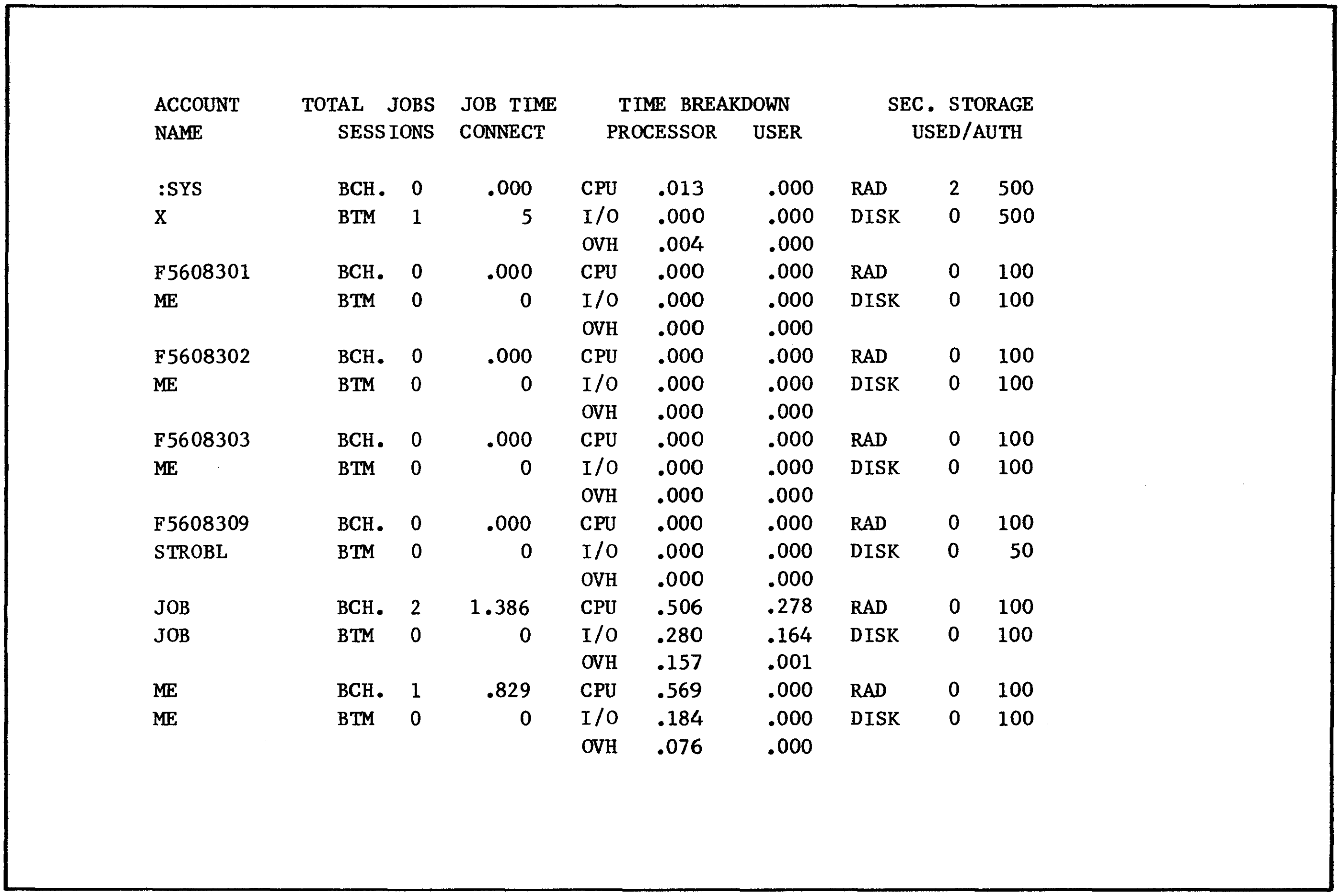
There are three forms of specification records available for the STATS option:

account causes statistics for users with the specified account to be summarized. A sample printout of a user's statistics resulting from a STAT control record is shown in Figure 3.

account,name causes only the statistics for individually named users in the account to be summarized.

zero-length record or !EOD (RET) causes all user statistics to be summarized. To use this feature, no other specification record may be previously encountered since entering the STATS mode.

To exit from the STATS mode, either a zero-length record (a (RET) character when using SUPER on-line) or an !EOD must be input.



```
ACCOUNT     TOTAL  JOBS  JOB TIME      TIME BREAKDOWN        SEC. STORAGE
NAME           SESSIONS  CONNECT     PROCESSOR    USER         USED/AUTH

:SYS           BCH.  0      .000   CPU    .013      .000   RAD      2   500
X              BTM   1         5   I/O    .000      .000   DISK     0   500
                                   OVH    .004      .000
F5608301       BCH.  0      .000   CPU    .000      .000   RAD      0   100
ME             BTM   0         0   I/O    .000      .000   DISK     0   100
                                   OVH    .000      .000
F5608302       BCH.  0      .000   CPU    .000      .000   RAD      0   100
ME             BTM   0         0   I/O    .000      .000   DISK     0   100
                                   OVH    .000      .000
F5608303       BCH.  0      .000   CPU    .000      .000   RAD      0   100
ME             BTM   0         0   I/O    .000      .000   DISK     0   100
                                   OVH    .000      .000
F5608309       BCH.  0      .000   CPU    .000      .000   RAD      0   100
STROBL         BTM   0         0   I/O    .000      .000   DISK     0    50
                                   OVH    .000      .000
JOB            BCH.  2     1.386   CPU    .506      .278   RAD      0   100
JOB            BTM   0         0   I/O    .280      .164   DISK     0   100
                                   OVH    .157      .001
ME             BCH.  1      .829   CPU    .569      .000   RAD      0   100
ME             BTM   0         0   I/O    .184      .000   DISK     0   100
                                   OVH    .076      .000
```

Figure 3. STAT Printout

DELSTATS This function causes user statistics to be deleted. The format of the DELSTATS record is

:D[ELSTATS]

≥option

⋮

The specification records for this option are identical to those used for STATS. For each user specified, the statistics are set to zero with the exception of the maximum RAD and/or disk pack space allowed, and the RAD and/or disk pack space used.

LIST This function causes a list of authorized users to be output. The format of the LIST record is

:L[IST][,account]

If account is present, all users validated for that account are listed; if account is not present, all users are listed. A sample printout is shown in Figure 4.

PASSWORD This function causes all passwords to be output. The format of the PASSWORD record is

:P[ASSWORD][,account]

The password for each file in theaccount is listed in EBCDIC if the password is printable; otherwise, it is listed in hexadecimal. If there is no password, the following message is printed:

```
****NONE****
```

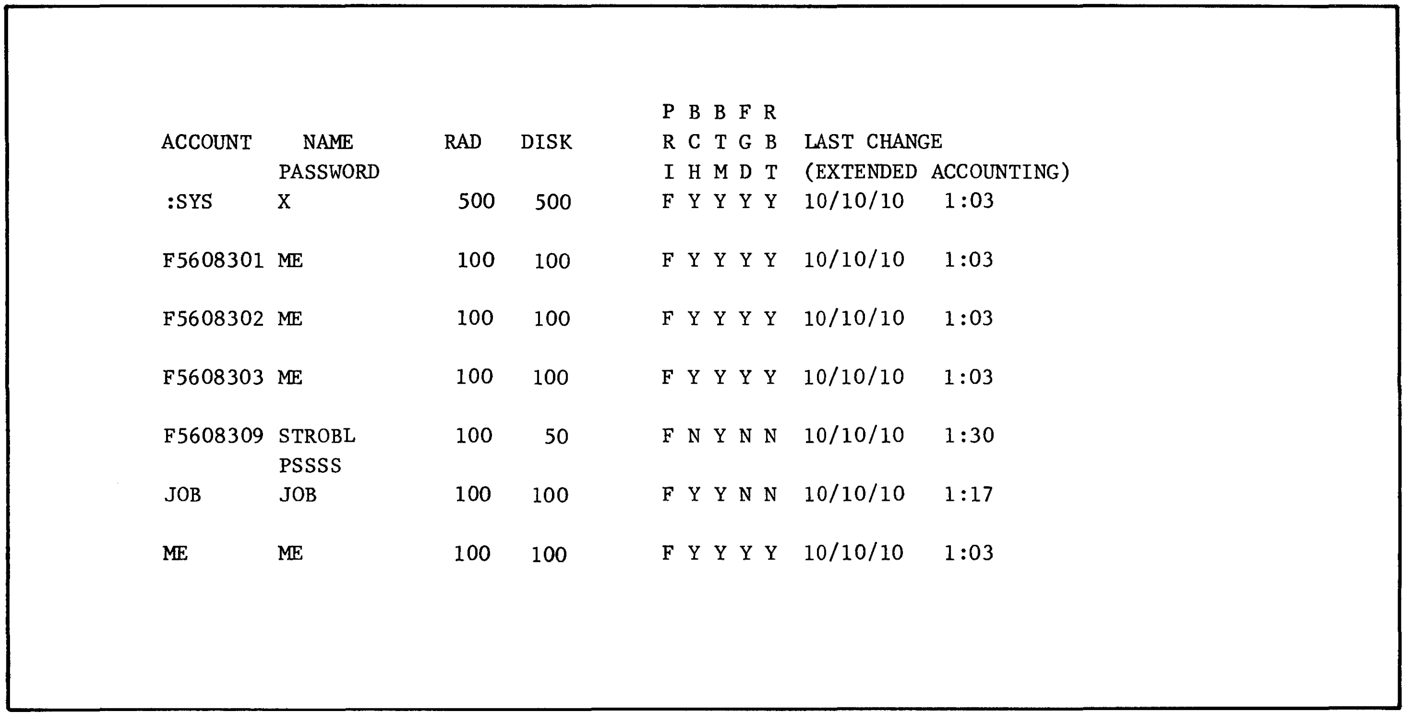
ERRORS AND ERROR MESSAGES

If the user makes an error when in batch mode, SUPER will print a dollar sign ($) under the field in error and output the message

```
SYNTAX ERROR, RECORD IGNORED
```

If the user makes an error when on-line, SUPER will print out the format required.

All other messages are self-explanatory.



```
                                          P B B F R
ACCOUNT    NAME       RAD   DISK          R C T G B  LAST CHANGE
         PASSWORD                         I H M D T  (EXTENDED ACCOUNTING)
:SYS     X            500    500          F Y Y Y Y  10/10/10    1:03

F5608301 ME           100    100          F Y Y Y Y  10/10/10    1:03

F5608302 ME           100    100          F Y Y Y Y  10/10/10    1:03

F5608303 ME           100    100          F Y Y Y Y  10/10/10    1:03

F5608309 STROBL       100     50          F N Y N N  10/10/10    1:30
         PSSSS
JOB      JOB          100    100          F Y Y N N  10/10/10    1:17

ME       ME           100    100          F Y Y Y Y  10/10/10    1:03
```

Figure 4. LIST Printout

4. USER ACCOUNTING

INTRODUCTION

Accounting for all BPM/BTM user activities that are subject to charges is implemented through Super and the accounting log (ACCTLG). Accounting data for a given job is contained in a control block called a Job Information Table (JIT for batch, AJIT for on-line, RJIT for real-time) along with a timer (TIMTEMP) and a number of accumulators that keep track of user and processor time.

At the end of each batch job and each terminal session an accounting record is written into a sequential Accounting Log file called :ACCTLG. Thus, the discrete record for the session is contained in the :ACCTLG file. The cumulative totals for the account up to the present time are contained in the :USERLG file (that was created by Super).

An accounting record is output at the end of a job. This record gives the total job time for a given job plus a breakdown of system services utilized, including secondary storage services (except private storage). Only nonzero accounting items are output on the user's summary.

In addition to the standard BPM/BTM accounting routines the system supplies entry points to optional, installation – designed routines if the system manager wants further control of accounting facilities. These optional accounting routines are implemented through use of the extended accounting parameter in the user's specification field. The installation routines are incorporated in the system by including them as ROMs during PASS1 of SYSGEN, and modifying the !LOCCT and !TREE cards (see Chapter 11).

ACCOUNTING FIELD FORMATS

To reduce user confusion, the formats of the BPM !JOB card and on-line LOGIN have been made as similar as possible. The format of the !JOB card is

```
!JOB  account [, name] [(extended accounting)]
      [priority] [, rbid] [. comments]
```

and the format for the on-line user is

```
!LOGIN:  account [, name] [(extended accounting)]
      [, password] (RET)
```

where

account is the account or project identifier. The "account" field must be at least one alphanumeric character but not more than eight. Batch jobs whose "account" field is omitted or exceeds eight characters will be aborted; on-line users will be asked to log in again.

name identifies the user. The name field is optional and, if it exists, will be appended to the "account" field to form the key for the authorized account search in :USERLG. If the name field is omitted, the account field only will be used for the key. Although the name field may consist of any number of characters for batch jobs, only the first 12 characters will be used (on-line users will have to log in again if the length is exceeded).

(extended accounting) is supplementary accounting information for the installation. The field is optional unless the installation accounting routines require it of users. If the extended accounting field exists, it must be enclosed in parentheses, but must not include blanks, embedded parentheses, periods, commas, or semicolons. For batch jobs, the first 24 characters, exclusive of the enclosing parentheses, are transferred to the JIT (on-line users will have to log in again if the field length exceeds 24 characters).

priority (for batch jobs only) specifies the priority of the job (0-F_{16}), where F is the highest (i.e., most urgent) priority. If no priority is specified, the default value is 1. A priority of 0 causes a job to be held in the job queue until the priority is changed by means of a !PRIORITY key-in. A priority specification is ignored (for purposes of scheduling) in a nonsymbiont system. If the priority exceeds the maximum authorized for the user, the job is aborted.

rbid (required only for remote batch operations) specifies a single hexadecimal digit identifying the remote terminal to receive output. To specify remote batch, priority (or two consecutive commas) must be specified. If rbid and priority are omitted, rbid is set equal to 0 (local device) and priority is assumed to be 1 (lowest).

password (for on-line only) is any password of one to eight characters. Only one password is legal for each unique name-account pair. However, the same password can be associated with several users and several names may be used with the same account.

ACCOUNT AUTHORIZATION

A batch user in BPM is subjected to the same account scrutiny as an on-line user in BTM. Two bits are reserved in each record of the :USERLG file so that users can be authorized for batch but not on-line and vice versa, with a third bit being reserved for foreground authorization.

Accumulated totals for both background and on-line system use are kept for the user in the authorization file. To keep symbiont and nonsymbiont systems functionally similar, the validity checking is performed by CCI, not the input symbiont.

When a job is selected for execution or a user logs in, a key is first constructed from the account and name fields from the !JOB card or LOGIN entry. A keyed read of :USERLG is performed using this key. If the key is invalid or a matching record does not exist, the job is aborted (if a batch job) or the user is requested to log in again (if an on-line user).

The format of the records in the :USERLG file is shown in Figure 5. The key is 21 bytes in length including the byte count, and consists of a 1-byte binary 20 and an 8-byte account number field, followed by a 12-byte name field. Both account number and name are EBCDIC fields left-justified in their respective fields, and are filled with trailing blanks. The account number must have at least one character but the name field may be all blanks. Neither name nor account may have embedded blanks or delimiters. The following five characters are the only restricted delimiters . , ; ( ). Both the extended accounting information and the LOGIN password are left-justified EBCDIC fields with trailing blanks. All other fields, except the flags in word 9 are positive binary integers. The date of the last update by SUPER (word 8) is the binary number which results from expressing the date as MMDDYY and converting this six-digit decimal number to binary; for example, 31st December 1999 would be 123199, which is 1E13F hexadecimal. All times are given in thousandths of a minute.

Further authorization checks may be made by installation routines by comparing the extended accounting field of a job against the authorization record (see "Supplementary Accounting Routines" in this chapter).

Accumulated totals for both background and on-line system use are also kept for the user in the authorization file. The totals are updated at the end of each job and terminal session. There is no way to distinguish how much of the accumulated total was contributed by batch jobs and how much by terminal sessions except for the total batch time which is kept separate from total connect time. However, the system manager may assign two different account numbers and authorize one for batch only and the other for on-line only to eliminate the problem. All the accumulated statistics are positive binary numbers, the times being in thousandths of a minute. The accumulated statistics may be reset by the use of the supervisory program SUPER. The maximum number of RAD and disk pack granules allowed for each job is obtained by subtracting the number used from the number allowed in the JIT during job initiation. If this number is exceeded during processing, the job is aborted.

The :USERLG file is generated by the Supervisory program (SUPER) rather than during initialization. If the file does not exist in the system (OPEN abnormal code X'03'), any user is considered authorized. If the file exists but the system returns an I/O error or abnormal code other than X'14', X'42', X'43' or X'55' when attempting to access the file, then only jobs whose account numbers are :SYS are allowed to run. This gives installation management the power to delete or repair the file without allowing general access to the system.

ACCOUNT TIMINGS

Accounting data pertaining to a job is kept in a control block called the Job Information Table (JIT). Job timings are handled in the following manner: a timer (TIMTEMP) and three accumulators (CEXT, IOTIME and OVHTIME) are kept in the JIT. TIMTEMP is incremented by one every thousandth of a minute. Each time there is a change of state the appropriate accumulator is incremented by the value in TIMTEMP, and TIMTEMP is reset to zero.

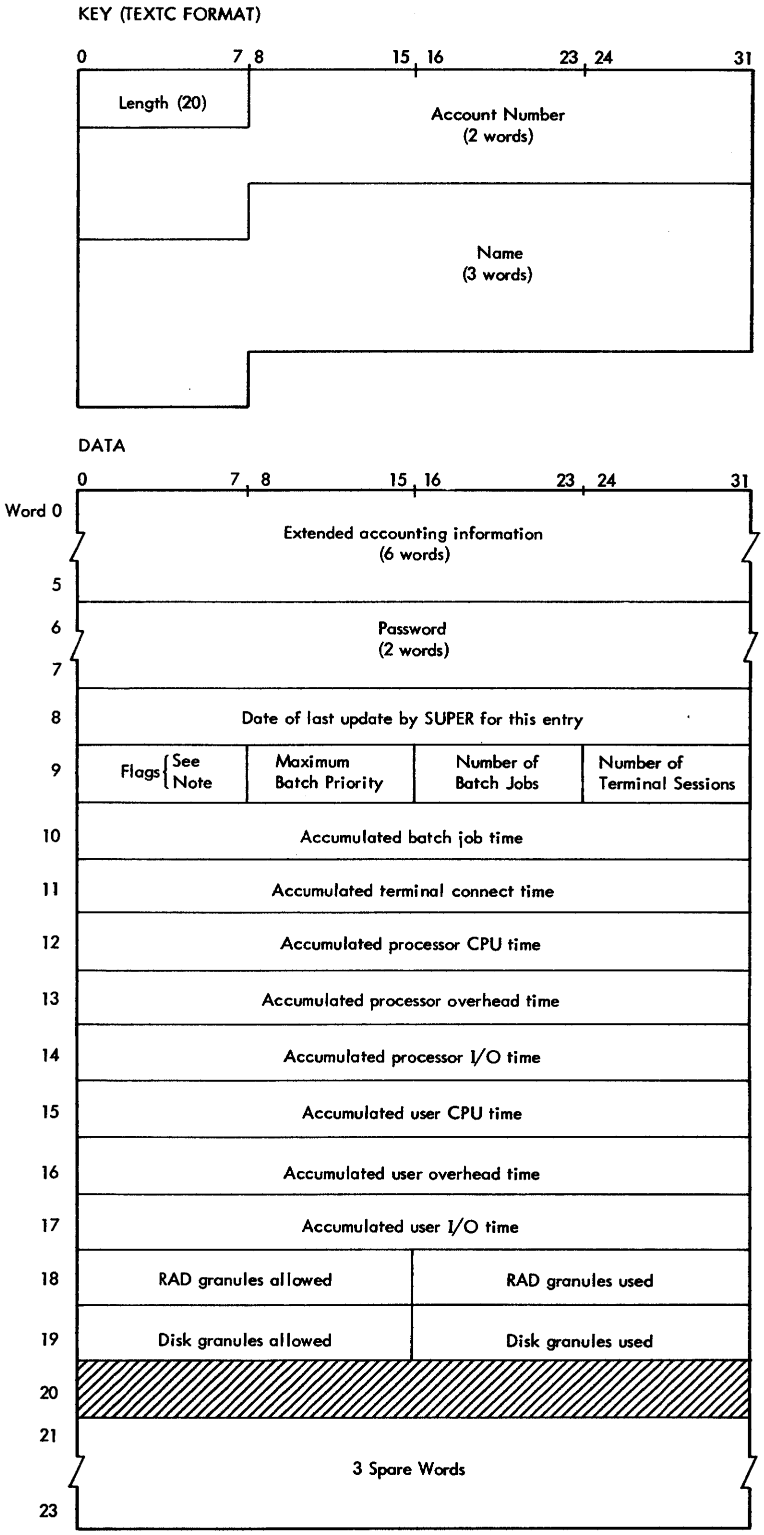
The appropriate accumulator is selected as follows: If a processor or user was in control of the CPU up to the change of state, the interval is considered execution time and CEXT is updated. If the Monitor was in control, the interval is overhead time and OVHTIME is updated. But if the Monitor was waiting for completion of an I/O operation for a user who requested a "wait", it is I/O time and IOTIME is updated. However, the Monitor time spent in preparing to issue the I/O operation is charged to overhead; I/O time is only the time spent waiting for completion of an I/O operation since the start I/O command. If the user did not specify "wait" either explicitly or implicitly, the I/O time is overlapped and not charged.

The JIT also contains three accumulators for processor time (TPEXT, TPOVT and TPIOT), and a matching set for user time (TUEXT, TUOVT and TUIOT). At the end of a job step, the appropriate set of these accumulators is incremented by the corresponding values in CEXT, OVHTIME and IOTIME, after it has been determined whether the program just terminated was a processor or a user program. Finally, at job end, the processor times are incremented by the amount of time used by CCI to terminate the job since the end of the last job step. The final values in the processor and user sets of accumulators are those that appear in the accounting record for the job.

The JIT fields ACCN (account number), UNAME (user's name) and EXTACC, are all left-justified, blank-filled EBCDIC values obtained from the !JOB card, or LOGIN message.

AUTHFLGS is a set of binary flags to indicate for which services the user is authorized.

BEGINDT and BEGINTM are the date and time at the beginning of the job. The date is the binary number which results from expressing the date as MMDDYY and converting this six-digit decimal number to binary; for example, 31st December 1999 would be 123199 which is 1E13F hexadecimal. The time is a binary number which is generated by taking the time-of-day presented as HHMM, multiplying the number of hours by 60, adding the minutes, multiplying the result by 1000 and converting this number to binary. Thus, it is the number of thousandths of a minute since the preceding midnight, expressed to the nearest minute. For example, 2:45 pm would be presented by the system as 1445,



KEY (TEXTC FORMAT)

0–7	8–15	16–23	24–31
Length (20)	Account Number (2 words)		
Name (3 words)			

DATA

Word	0–7	8–15	16–23	24–31
0–5	Extended accounting information (6 words)			
6–7	Password (2 words)			
8	Date of last update by SUPER for this entry			
9	Flags {See Note	Maximum Batch Priority	Number of Batch Jobs	Number of Terminal Sessions
10	Accumulated batch job time			
11	Accumulated terminal connect time			
12	Accumulated processor CPU time			
13	Accumulated processor overhead time			
14	Accumulated processor I/O time			
15	Accumulated user CPU time			
16	Accumulated user overhead time			
17	Accumulated user I/O time			
18	RAD granules allowed		RAD granules used	
19	Disk granules allowed		Disk granules used	
20	▨			
21–23	3 Spare Words			

where in Word 9

Flag Bit 0	indicates authorization to run batch jobs.
Flag Bit 1	indicates authorization to run on-line.
Flag Bit 2	indicates authorization to run foreground programs.
Flag Bit 3	indicates authorization to use remote batch.
Flag Bits 4-7	are spare.

Figure 5. Authorization File Record (:USERLG)

which is 885000 thousandths of a minute since midnight and which would appear in the accounting record as D8108 hexadecimal. The reason for expressing the time in thousandths of a minute (which are always zero) is for compatibility with the other accounting timings and so that no format changes will be necessary if time-of-day is ever modified to include thousandths of a minute.

The JIT field LINEND (line number) is set to X'FF' for batch jobs. MAXPRT is the maximum permissible priority, which is obtained from the :USERLG file. The remaining accounting fields in the JIT through word 26 can be deciphered by using the following glossary:

TMP	=	Temporary
PRM	=	Permanent
DC	=	RAD
DP	=	Disk Pack
JB	=	Job
MX	=	Maximum
PK	=	Peak
TP	=	Tape
Access	=	I/O accesses
MNTS	=	Mounts

BATCH PRIORITY DEFAULT LIMITS

The priority on the !JOB control command is used to obtain the appropriate default limits for each of the job priorities (0 through F) and insert them in the Job Information Table. A separate set of default limits is provided for each priority so that a !LIMIT card is required only for jobs whose limits differ from the defaults for that particulary priority. This is true for both symbiont and nonsymbiont systems.

The separate set of default limits for each of the job priorities 0 through F are in the M:DLIMIT module. Limit fields in the JIT are initialized with defaults from M:DLIMIT according to the priority of the job; that is, during !JOB card processing, M:JOBR will use the priority on the !JOB card to obtain the appropriate job default limits from M:DLIMIT and insert them in the JIT. The limit fields of the JIT are

FPOOL size and number.

IPOOL size and number.

Maximum number of punched cards.

Maximum number of processor pages.

Maximum number of user pages.

Maximum number of diagnostic pages.

Maximum execution time (in minutes).

Maximum number of scratch tapes.

Maximum temporary direct access granules.

Maximum permanent direct access granules.

The values in the JIT are modified by M:LIMR if a !LIMIT command is encountered. The limits specified on the !LIMIT card overlay those defaults in the JIT. However, the limits may be only made smaller than the defaults, not larger. The limits for PSTORE are also constrained by the value in the :USERLG file.

In the event that the priority is changed by the operator, the new priority is used for job scheduling but the old priority (that on the !JOB command) is used for selecting default limits.

There are 16 parallel default limit tables, one for each priority, and the contents of these tables are set during PASS2 of SYSGEN via the :DLIMIT control command which defines the specification of different default limits for each of the priorities.

ACCOUNTING LOG FILE

A discrete accounting record for each batch job and each on-line terminal session is written into a common Accounting Log file (:ACCTLG) in the :SYS account. Each record contains an identifier so that batch records are distinguishable from on-line records (byte 0 of word 12 contains the line number of the BTM user or X'FF' for a batch job). The format of the records in :ACCTLG is shown in Figure 6.

The account number (word 0-1), the name (words 2-4), the extended accounting information (words 5-10), and the system ID (word 13) are left-justified, blank-filled EBCDIC fields. The run status (word 12) and the job origin (word 13) are collections of binary flags.

The remaining fields in the record are all binary integers. Of these, all are positive values with two exceptions. The line number (word 12) for a batch job will be set to X'FF', and the number of permanent RAD granules (word 27) and permanent disk pack granules (word 28) may be a two's-complement negative binary half-word integer if the job released more permanent granules than it obtained.

The date (word 14) is the binary number that results from expressing the date as MMDDYY and converting this six-digit decimal number to binary; for example, 31st December 1999 would be 123199, which is 1E13F hexadecimal.

The start and end time-of-day (words 15 and 16) are binary numbers generated by taking the time-of-day presented as HHMM, multiplying the number of hours by 60, adding the minutes, multiplying the result by 1000 and converting this number to binary. Thus, it is the number of thousands of a

	0 – 7	8 – 15	16 – 23	24 – 31
0	Account Number (2 words)			
2 4	Name (3 words)			
10	Extended Accounting Information (6 words)			
11	▨	▨	▨	▨
12	Line number (or batch X'FF')	Priority	Final run status	Number of steps or subsystems
13	Job origin (Batch, RJE, BPM,...)	System ID in EBCDIC (e.g., F00)		
14	Date (at job or terminal session start)			
15	Job start or LOGIN time			
16	Job end or LOGOFF time			
17	Number of characters transmitted (zero for batch)			
18	Terminal active time (or accumulated batch quanta)			
19	Job sequence number		Number of disk pack mounts	▨
20	Number of cards read		Number of cards punched	
21	Number of processor payes		Number of user pages	
22	Number of diagnostic pages		Number of save tapes	Number of scratch tapes
23	Number of tape accesses			
24	Number of RAD accesses			
25	Number of disk accesses			
26	▨	▨	▨	▨
27	Number of TEMP RAD granules		Number of PERM RAD granules	
28	Number of TEMP disk granules		Number of PERM disk granules	
29	▨	▨	▨	▨
30	Processor execution time			
31	Processor overhead time			
32	Processor I/O time			
33	User execution time			
34	User overhead time			
35	User I/O time			
36 39	4 spare words for expansion			

Figure 6. Accounting Log Record Format (:ACCTLG)

minute since the preceding midnight, expressed to the nearest minute; for example, 2:45 p.m. would be presented by the system as 1445 which is 885000 thousandths of a minute since midnight and which would appear in the accounting record as D8108 hexadecimal. The reason for expressing the time in thousandths of a minute (which are always zero) is for compatibility with the other accounting timings and so that no format changes will be necessary if time-of-day is ever modified to include thousandths of a minute.

All lengths of time in the accounting record (words 18, and 30 through 35) are given in thousandths of a minute.

If the :ACCTLG file is deleted during job processing, it has no effect on the operation of the system. If the :ACCTLG file does not exist at end of job or terminal session, a new file is created. If it does exist, it is positioned to end-of-file. In either case, the accounting record is written and the file is closed.

SUPPLEMENTARY ACCOUNTING ROUTINES

This feature provides the manager of a BPM/BTM installation further control of accounting facilities. The system manager may modify the extended accounting field which was transferred from the JIT or any other fields in the accounting record; he may write his own accounting log in addition to or instead of :ACCTLG, omit selected records, or take any other action that suits his purpose. The supplementary accounting feature consists of an extended accounting field on the !JOB card, in the JIT, the :USERLG file, the :ACCTLG file, and an exit to installation routines at job start in M:JOBR and at job end in M:ENDJOB. For on-line sessions, the same routines are entered from the BTM Executive during LOGIN and BYE. The installation-supplied batch accounting routine must be included when forming CCI during SYSGEN; the on-line version must be included in the root of the Monitor.

The entry points for batch and on-line have the same name (M:ACINIT for initialization and M:ACTERM for termination) and may or may not be the same routines, at the discretion of the installation.

The extended accounting field on the !JOB card or LOGIN message response is optional unless required by the installation. It is enclosed in parentheses and follows the name field as if it were a subscript. The extended accounting field may be any of any length but must not include blanks, parentheses, periods, commas, or semicolons. The first 24 characters, exclusive of the enclosing parentheses, are transferred to the JIT. After the authorization checks are made, an exit is made to the installation job initiation accounting routine. The exit is made via an SREF (secondary reference), so if there is no installation routine, it becomes a NOP (no-operation).

The installation may supply a batch job and/or on-line initiation accounting routine whose name must be M:ACINIT, and a batch and/or on-line termination routine whose name must be M:ACTERM. Exits will be made to these names, which will be assembled as SREFs at the following places:

1. M:ACINIT Batch job initiation in M:JOBR (CN704905), and on-line log-in in the BTM EXEC (CN705415).

2. M:ACTERM Batch termination in M:ENDJOB (CN704909) and on-line log-off in the BTM EXEC (CN705415).

In each case, exit to the installation accounting routine is via BAL on register 15. Register 3 contains the address of the authorization record for M:ACINIT and the accounting log record for M:ACTERM, and register 5 contains the address of the initialized JIT/AJIT. JIT/AJIT can also be addressed directly via the symbol JIT, or indirectly via location X'4F' or the symbol CJOB. The JIT is identified by the fact that bit 0 of word 0 is set to 0; AJIT has bit 0 of word 0 set to 1.

Upon return from the installation initiation routine, the job will be aborted if register 3 has been set to 0; an on-line user will be asked to log in again. This gives the system manager an opportunity to police the job or modify the JIT, based on information in the extended accounting field.

At job end, the information in the JIT is transferred to the :ACCTLG file. The conditions are identical to the job initiation exit except that register 3 points to the accounting record (if the installation sets register 3 to 0, writing of the accounting record in :ACCTLG is suppressed).

If the M:ACTERM routine exists, it will receive control after the record is built in core but before the record is written in the :ACCTLG file. M:ACINIT can abort a job and M:ACTERM can suppress writing of the accounting record if necessary.

Caution: All registers except register 3 are expected to remain intact. Since the installation accounting routine becomes part of the CCI (for batch) or the Monitor (for on-line) which operates in the master mode, a coding error or altered register contents could result in a system crash.

An installation wishing to supplement the standard BPM/BTM accounting includes ROMs with the entry names M:ACINIT and M:ACTERM during PASS1 of SYSGEN, and modifies the LOCCT and TREE table.

ACCOUNTING SECONDARY STORAGE

Storage limits are checked separately for permanent and temporary storage. Private storage does not enter into limit check or accounting of space used. Accounts are kept for

1. Accumulated public storage on RAD or disk pack devices.

2. Number of accesses to RAD, disk pack, and magnetic tape.

3. Number of private volumes mounted, magnetic tape or disk pack.

All nonzero accounting items are printed on the user's summary.

ACCOUNTING SHEET INFORMATION

The items given in Figure 7 below are all the accounting messages that can appear on the final page of the BPM/BTM printed output. Printing is suppressed for any item whose associated value is zero; that is, if a user does not mount any magnetic tapes during his job, the entry "# OF TAPE MOUNTS" will not be printed, as opposed to a message "# OF TAPE MOUNT 0".

TOTAL JOB TIME	Less than or equal to the sum of the next six entries that follow below it; the possible discrepency being the noncharged short quanta used while the system is waiting for a magnetic tape or disk pack to be mounted or the symbiont to catch up.
PROCESSOR EXECUTION TIME PROCESSOR I/O TIME PROCESSOR OVERHEAD TIME USER EXECUTION TIME USER I/O TIME USER OVERHEAD TIME	All time is expressed as XXX.YYY where X = minutes Y = thousandths of a minute
# OF CARDS READ	Includes the !JOB card and all data cards and control cards except the !FIN card.
# OF CARD PUNCHED	Includes ID, !JOB, !BIN, and !EOD cards, but not the blank card inserted by the symbiont between jobs. If there is no punched output from the job, the !JOB and ID cards are not punched and the "# OF CARDS PUNCHED" line is omitted.
# OF PROCESSOR PAGES OUT	Includes all printed output generated by processors such as CCI, PASS1, FORTRAN, METASYM, etc., plus the two ID pages at the beginning of the job and the accounting page at the end. Therefore, the entry is never zero.
# OF USER PAGES OUT	Includes number of pages of user output. Since the CCI processor and the user print on the same page, the page containing the !RUN command will not be counted as a user output page.
# OF DIAGNOSTIC PAGES OUT	Number of pages output through the M:DO DCB to a symbiont file (or to a printer in a nonsymbiont system), including core dumps, snaps, and debug outputs. Since the total will be included in the user pages count when user and diagnostic output goes to the same device, the message will usually not be printed. Additionally, if the output goes to a user file, it will also not be counted even though it goes through M:DO.
PERM RAD GRANULES USED PERM DISC GRANULES USED	PERM granules used is the net change in the number of permanent granules on the specified device type that are affected by the job. These values can be negative.
TEMP RAD GRANULES USED TEMP DISC GRANULES USED	TEMP granules used is the peak value for the number of temporary granules on the specified type used during the job.
# OF TAPE ACCESSES # OF RAD ACCESSES # OF DISC ACCESSES	Includes reads, writes, and file positioning accesses on the specified device type. Seeks are not counted as they are considered part of a read or write. A chargable access is actually a request to queue.
# OF TAPE MOUNTS # OF DISC MOUNTS	Includes all mounts of the specified device type unless the device is premounted by the operator.

Figure 7. User Accounting Summary Messages

5. BTM PERFORMANCE MONITOR

INTRODUCTION

The BTM Performance Monitor is a system management tool that accumulates critical statistics during system operation. These statistics measure the performance of BTM in a given hardware configuration and provide a profile of the current user environment. The environment in any given facility will be in a state of fairly continuous change that reflects variations in the user load on a moment-by-moment basis.

The various statistical reports are received on-line and enable the system manager to continuously monitor the user stream. Control over the Report Generator is obtained by selecting or rejecting option queries output by the Performance Monitor. These reports, in turn, supply the data necessary to optimize performance through system tuning.

The Performance Monitor consists of two groups of routines: the first is resident and performs the data collection, and the second generates the subsequent reports.

DATA COLLECTION

Data is collected in different tables whenever an activation character is received from a user's console or when a user quantum is dismissed by the BTM Executive. Clocks 2 and 3 (500 Hz) are used by the Performance Monitor.

REPORT GENERATION

A Meta-Symbol routine copies the tables containing the collected data into the common storage of a FORTRAN Report Generator that performs the following functions:

- Reports statistics since the beginning of the accumulation of statistics (system reset).
- Reports statistics with reference to a base file created at a time after the accumulation of statistics was started (after system reset).
- Reports "snapshot" statistics and summary statistics from a history file.
- Creates a base file as a reference point.
- Creates a series of records in a file periodically to create a history file.
- Summarizes the batch accounting log during history file creation since base file creation or system reset.
- Lists quantum statistics separately for each of the two on-line quantum levels (q_1 and q_2).
- Lists snapshot summaries chronologically and then sorts and relists the summaries by user intensity and users logged on.
- Provides accounting for short, self-dismissed quanta.
- Provides subsystem listings that include the percentage of on-line time and the percentage of tasks.
- Identifies by name the subsystems causing maximum quantum extensions and the number of RAD and disk pack accesses that caused the extension.

INITIALIZATION AND TERMINATION

The accumulation of statistical history is a discrete function of time and is a proprietary service. Initialization and termination are controlled by the key-ins

!PMS (RET)

!PMX (RET)

where

PMS specifies to start accumulating statistics.

PMX specifies to stop accumulating statistics (default).

BTM SCHEDULING

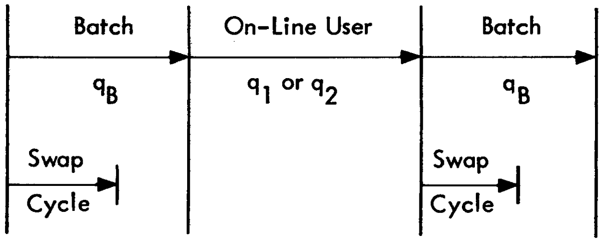
Interpretation of the BTM statistics requires a basic understanding of the BTM time-sharing algorithm. BTM uses a simple procedure that assigns slices of system time, called quanta, alternately to the batch stream and to one of the on-line users. This procedure is demonstrated in Figure 8.

QUANTA

The batch quantum has a preset value that is set either at SYSGEN time or modified at run time via a key-in. The batch quantum is used by the time-sharing system to swap the data area of the last on-line user out of core and to swap the procedure and data area of the next on-line user into core. The swap cycle includes swap out and swap in, and is overlapped with the execution of the batch job.

The batch quantum is terminated if the swap cycle is completed, the batch preset q_B has been reached, and no file I/O write action is pending. If an I/O write operation has begun, the batch quantum is extended until the I/O operation has been completed.

The on-line quantum will be assigned a first-level preset value, q_1, if it is the first quantum of the respective on-line task. The on-line quantum is assigned a second-level preset value, q_2, if the respective on-line task has already received one quantum. An on-line quantum is dismissed



Batch Quantum

1. Never smaller than swap cycle.
2. Extended past q_B preset for I/O completion.

On-Line Quantum

1. Dismissed early if:
   a. Program requested.
   b. Task completed.
2. Extended past preset to complete I/O action.

Figure 8. BTM Time Sharing

early if the task is completed or if the task requests dismissal via CAL1,8 X'20'.

On-line quanta may be extended past the preset value only if a file I/O action is pending. In this case the on-line quantum will be extended until the I/O operation is completed. Extensions are likely to occur in on-line tasks since multiple file accesses are common in the on-line subsystems.

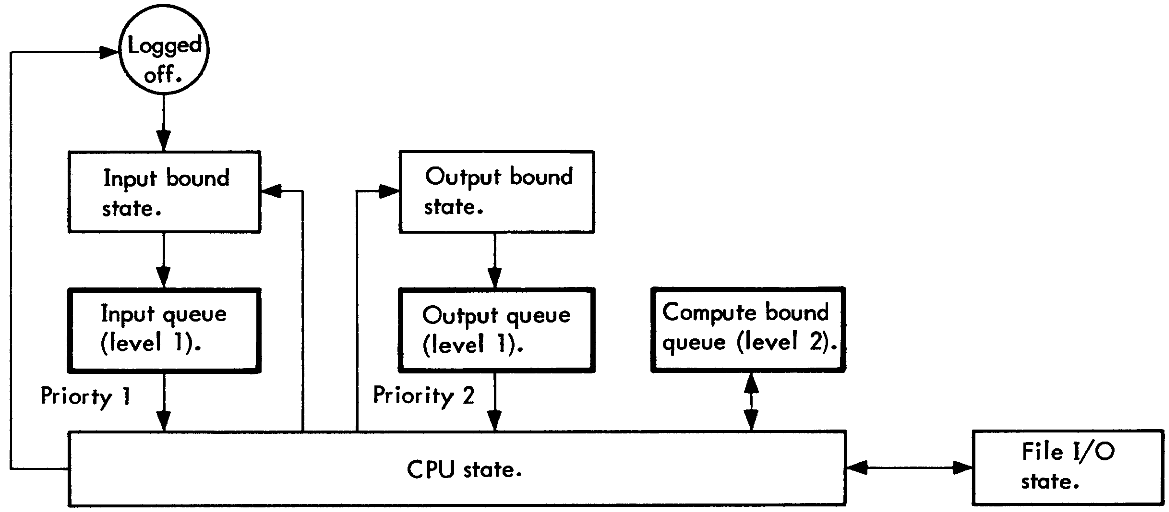
USER STATES

On-line users migrate through identified user states under the control of the scheduling algorithm. The user state diagram is shown in Figure 9. There are three queues shown on the state diagram: the input queue, the output queue, and the compute queue. The scheduler allows only one user at a time to receive CPU and I/O time. The on-line quantum begins when a user enters the CPU or file I/O states and terminates when the user leaves the CPU state. The user is then either queued for further service or placed in the input bound or output bound state.

The user migrates to the input bound state when prompted by the BTM system. The user state will change from the input bound state to the input queue when the first activation character is received in the input buffer.[†] If the user is in the executive level of execution, each character is considered an activation character. If the user is at the subsystem or user level of execution, the activation character is usually a carriage return. A user entering the input queue has priority for scheduler service over users in the other two queues. Such a user is granted a quantum q_1 when he gains service.

When a user receiving CPU service fills the output buffer,[†] the user's quantum is dismissed, the task is terminated, and the user state becomes output bound. An output bound user will remain in the output bound state until the character count indicates less than 10 characters remain in the output buffer. When this occurs the output bound user enters the output queue, which receives service of duration q_1 when the input queue is empty.

[†] The input buffer size and output buffer size are defined at SYSGEN time, using the IBUFSIZE and OBUFSIZE options respectively. The default size for both buffers is 100 characters



An on-line quantum begins when a user enters the CPU state and terminates when the user leaves the CPU state.

Figure 9. BTM User States

When a user receiving CPU service consumes the on-line quantum without completing the task, the user is considered compute bound and is entered into the compute bound queue. The compute bound queue receives service only when the input queue and the output queue are both empty; users queued at this state receive a quantum q_2 each time they are serviced.

SCHEDULING CONTROL

The installation manager may control the operation of the scheduler by assigning appropriate values to the quantum presets received by the user queues. The input queue and the output queue receive a first level quantum q_1, which is chosen such that 80 percent to 90 percent of the user tasks may be completed in one on-line quantum.

The users in the compute bound queue have received one first level quantum and are sched' led for a second level quantum q_2 only after the input and output queues are empty. Since the compute bound user receives secondary service, it is possible to increase q_2 as a device for reducing the number of scheduling cycles required to process each compute bound task. Increasing q_2 for heavier user loads has the effect of increasing the percentage of the system available to the on-line users.

Once BTM scheduling considerations are understood, the primary measures of performance can be addressed.

BASIC STATISTICS

The statistics below are the most important measures of the performance of the system. Others will become more meaningful and useful with increased knowledge and experience with BTMPM.

Statistic	Information Given
Time Sample	What interval is included in report?
Users Logged	How many users on the system?
Intensity	How heavy is the user load?
Interactive Response	How long does it take to receive and complete one quantum?
ETMF	How many seconds must a user wait for one second of system time?
On-Line Time Percentage	How much of the system is required by on-line users?
Interactive Tasks Percentage	How many of the tasks are completed with only one quantum of preferential service?

USING THE REPORT GENERATOR

The flow chart in Figure 10 provides an overall view of the flow of the Report Generator. Assuming the load module for the Report Generator is called 'STAT', the BTMPM may be accessed as follows:

```
!RUN
LOAD MODULE FID:STAT
;G
```

The RUN subsystem will then start executing STAT. At the beginning of execution, some consistency checks are performed. Any error detected causes printing of the appropriate error message and control is returned to the BTM Executive. The error messages are as follows:

```
BTMPM TABLES AND/OR LOAD MODULE NONEXISTENT
```

Either the tables created by SYSGEN (:BTM card) or the resident portion of STATISTICS package is not present.

```
DIMENSIONS (COMMONS) .LT. DIMENSIONS (SYSGEN)
```

BTMPM has some COMMONs dimensioned with the maximum default values for the options on the SYSGEN :BTM card. If the maximum limits have been altered, the dimension of the COMMONs should also be altered in the nonresident portion of memory.

```
BTM PERFORMANCE MONITOR IS NOT OPERATIONAL
```

The Performance Monitor has not been initiated. The message can also appear any time during execution if the PMX key-in is issued by the computer operator.

If there are no errors, the following header is printed:

```
*BTM PERFORMANCE MONITOR***

REPLY WITH Y OR N OR X AND
CARRIAGE RETURN
```

BASE FILE CREATION

The Report Generator then asks the first question about report statistics. A "Y" answer directs the execution to the report writers. A "N" answer causes a branch to the file

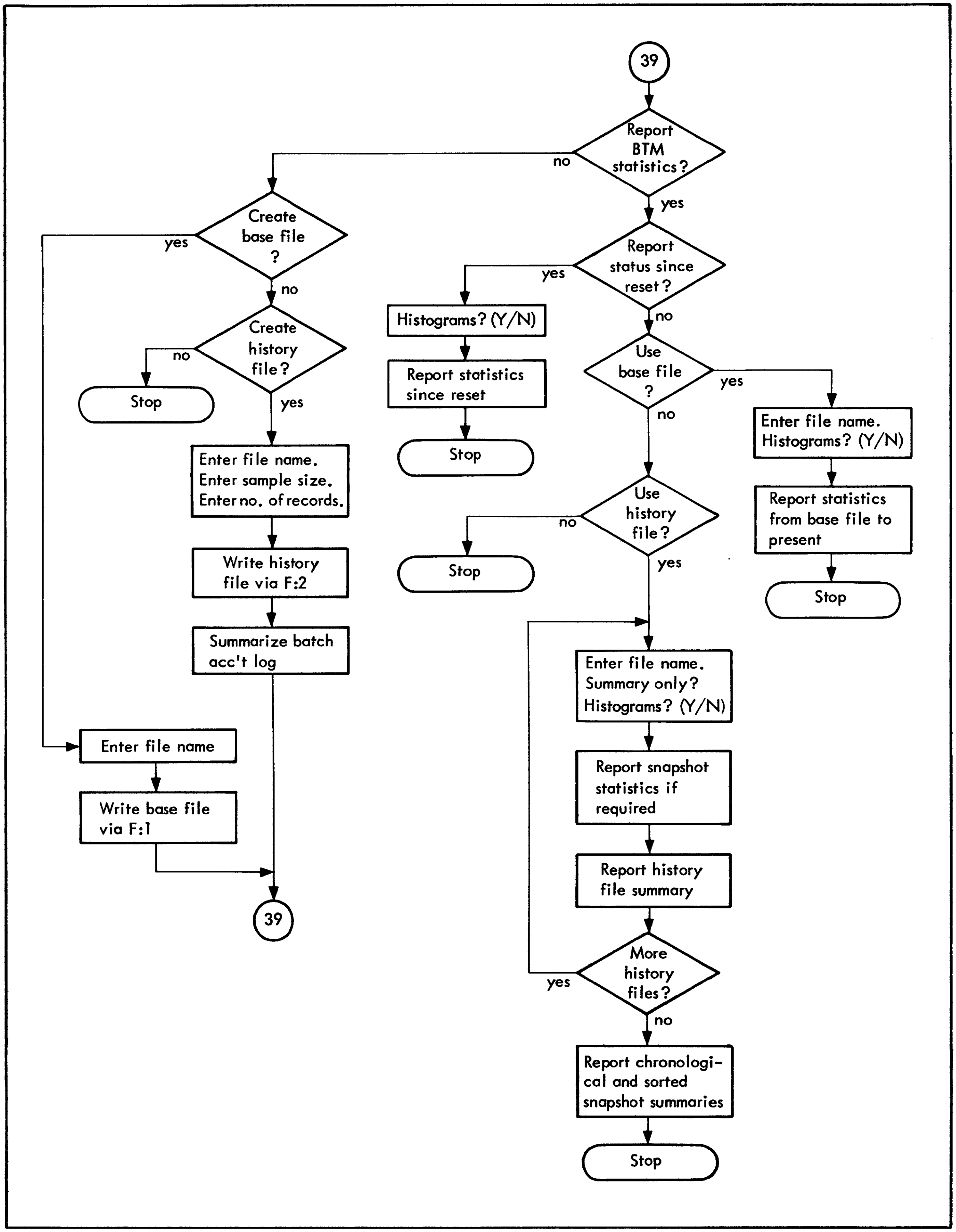

39
Report BTM statistics?
no
yes
Create base file ?
yes
no
Create history file?
no
Stop
yes
Enter file name. Enter sample size. Enter no. of records.
Write history file via F:2
Summarize batch acc't log
Enter file name
Write base file via F:1
39
Report status since reset?
yes
Histograms? (Y/N)
Report statistics since reset
Stop
no
Use base file ?
yes
Enter file name. Histograms? (Y/N)
Report statistics from base file to present
Stop
no
Use history file?
no
Stop
yes
Enter file name. Summary only? Histograms? (Y/N)
Report snapshot statistics if required
Report history file summary
More history files?
yes
no
Report chronological and sorted snapshot summaries
Stop

Figure 10. BTMPM Report Generator Flow Chart

creation portion of the program. Example 1, below assumes an "N" answer first.

The program will then ask if the base file is to be created. A "Y" response to the base file question causes the program to ask for the name of the file. The first eight characters will be used to name the file that contains the statistical status of BTM at that moment. A message confirming file creation is typed after the file is created. The program then resumes questioning from the beginning to allow a user immediate use of the base file by asking for a report referencing that file.

If the file already exists with the same name, the option to write over the old file is given to the operator by the message

```
FILE EXISTS. PROCEED?
```

A "Y" answer causes the old file to be written over. If the answer is "N", the operator will receive the file name query again. See Examples 2 and 3.

HISTORY FILE CREATION

An "N" response to the base file question results in a history file question. An "N" response to the history file question stops execution. A "Y" response causes history file questions to be typed as shown in Example 4.

The history file name may be entered by the operator and may be from 1 to 8 characters in length. If the history file name is the same as an already existent file name, the following message will be typed:

```
FILE EXISTS. PROCEED?
```

A "Y" reply causes the new file to replace the old; an "N" reply causes the file name query to be repeated. See Examples 5 and 6. The sample size must be a number from 01 to 99 to satisfy the FORTRAN format requirement. The sample size determines how often the history file records are to be created. In the above example, the number of records defines an 8-hour record of BTM with 15-minute snapshots.

The Report Generator creates files periodically until the specified number of records has been created. The Report Generator then reads the Batch Accounting Log and prints a summary of the batch jobs completed during the creation of the history file. This summary is useful since the batch activity has a pronounced effect on the on-line performance.

Example 1. Base File Creation with "N" Answer

```
   REPORT BTM STATISTICS ?
 —→?N(RET)
   CREATE BASE FILE ?
   ?Y(RET)
   ENTER FILE NAME BELOW.
   ?FILEBASE(RET)
   BASE FILE FILEBASE CREATED AT 09:41 NOV 16,'70
```

Example 2. Create a New Base File (Save Old Base File)

```
   REPORT BTM STATISTICS ?
   ?N(RET)
   CREATE BASE FILE ?
   ?Y(RET)
   ENTER FILE NAME BELOW.
   ?FILEBASE(RET)
   FILE EXISTS.  PROCEED?
 —→?N(RET)
   ENTER FILE NAME BELOW.
   ?FILEBAS1(RET)
   BASE FILE FILEBAS1 CREATED AT 09:43 NOV 16,'70
```

Example 3. Create a New Base File (Overwrite Old File)

```
   REPORT BTM STATISTICS ?
  ?N(RET)
   CREATE BASE FILE ?
  ?Y(RET)
   ENTER FILE NAME BELOW.
  ?FILEBASE (RET)
   FILE EXISTS.  PROCEED?
→?Y (RET)
   BASE FILE FILEBASE CREATED AT 09:44 NOV 16,'70
```

Example 4. Create a History File

```
   REPORT BTM STATISTICS ?
  ?N (RET)
   CREATE BASE FILE ?
  ?N(RET)
   CREATE HISTORY FILE?
→?Y (RET)
   ENTER FILE NAME BELOW.
  ?FILEHIST (RET)
   ENTER SAMPLE SIZE, 01 TO 99, MINUTES.
  ?15(RET)
   ENTER # OF RECORDS, 01 TO 99.
  ?33(RET)
```

Example 5. Create a New History File (Save Old File)

```
   REPORT BTM STATISTICS ?
  ?N(RET)
   CREATE BASE FILE ?
  ?N(RET)
   CREATE HISTORY FILE ?
  ?Y(RET)
   ENTER FILE NAME BELOW.
  ?FILEHIST(RET)
   FILE EXISTS.  PROCEED?
→?N(RET)
   ENTER FILE NAME BELOW.
→?FILEHIST (RET)
   ENTER SAMPLE SIZE, 01 TO 99, MINUTES.
  ?11 (RET)
   ENTER # OF RECORDS, 01 TO 99.
  ?22 (RET)
```

Example 6. Create a New History File
(Overwrite Old File)

```
    REPORT BTM STATISTICS ?
   ?N(RET)
    CREATE BASE FILE ?
   ?N(RET)
    CREATE HISTORY FILE ?
   ?Y(RET)
    ENTER FILE NAME BELOW.
   ?FILEHIST(RET)
    FILE EXISTS.  PROCEED?
-->?Y(RET)
    ENTER SAMPLE SIZE, 01 TO 99, MINUTES.
   ?12(RET)
    ENTER # OF RECORD, 01 TO 99.
   ?24(RET)
```

BATCH ACCOUNTING LOG SUMMARY

The Batch Accounting Log Summary is printed when the history file is created. As shown in the format and sample data in Figure 11, the Batch Accounting Log lists jobs according to their origin. The TOTAL indicates the sum of LOCAL jobs (submitted over the counter) and those of REMOTE origin. If there were abnormal jobs, their total is also indicated according to their origin.

Under the TIMINGS AND PERCENTAGES heading, the information is broken down in three fields: SYSTEM, LOCAL, and REMOTE. LOCAL and REMOTE correspond to the job origin of the entries (EXECUTION, OVERHEAD, and I/O); the TOTALs (expressed in minutes) are the accumulated processor and user values (expressed in percentages). The information under the SYSTEM field has the same general format.

The accounting information for peripherals follows the same logic.

```
                                          <- JOB ORIGIN ->
                                   TOTAL   LOCAL    REMOTE
                   NORMAL JOBS      200     180       20
                   ABN. JOBS          4     75.0 %    25.0 %

                          * TIMINGS & PERCENTAGES *

                <--- SYSTEM ----> <---- LOCAL ----> <---- REMOTE ---->
                TOTAL  LOCAL REM   TOTAL  PROC  USER  TOTAL PROC  USER

     EXECUTION  200.0  75.0  25.0  150.0  33.3  66.7   50.0  40.0  60.0
     OVERHEAD   150.0  66.6  33.4  100.0  55.0  45.0   50.0  37.0  63.0
     I/O        100.0  20.0  80.0   20.0  70.0  30.0   80.0  30.0  70.0

                               TOTAL     LOCAL     REMOTE

     CARDS (READ               1534     60.0 %    40.0 %
           (PUNCHED              20     60.0 %    40.0 %

           (PROCESSOR           334     60.0 %    40.0 %
     PAGES (USER                 25     60.0 %    40.0 %
           (DIAGNOSTIC            0     60.0 %    40.0 %

     TAPES (SCRATCH              10     60.0 %    40.0 %
           (SAVED                 2     60.0 %    40.0 %

              (TAPE            2431     60.0 %    40.0 %
     ACCESSES (RAD             2200     60.0 %    40.0 %
              (DISC            3334     60.0 %    40.0 %

     TEMPORARY GRANULES (DISC  340      60.0 %    40.0 %
                        (RAD    30      60.0 %    40.0 %

     PERMANENT GRANULES (DISC  250      60.0 %    40.0 %
                        (RAD    30      60.0 %    40.0 %

     DISC PACK MOUNTS            6      60.0 %    40.0 %
```

Figure 11. Batch Accounting Log Summary Example

REPORT STATUS SINCE RESET

Assume that a user has either just entered the Report Generator or has just created a history file or base file. This user would reply with a "Y" to the first question. A "Y" response to the second question would allow the user to get a report that covered the time from reset until the present time.

Example:

```
   REPORT BTM STATISTICS ?
  ?Y(RET)
   REPORT STATUS SINCE RESET ?
→?Y(RET)
   ARE HISTOGRAMS DESIRED?
  ?Y(RET)
```

The report covering the period from reset to present would then follow.

REPORT USING BASE FILE

An "N" response to the second question causes a query to determine if a base file is to be used.

Example:

```
   REPORT BTM STATISTICS ?
  ?Y(RET)
   REPORT STATUS SINCE RESET ?
  ?N(RET)
   USER BASE FILE ?
→?Y(RET)
   ENTER FILE NAME BELOW.
  ?FILEBASE(RET)
   ARE HISTOGRAMS DESIRED?
  ?Y(RET)
```

The report covering the period from base file creation to present would then follow.

If a base file was created and the user answers the base file question with a "Y", the program will ask for the name of the file and whether histograms are desired in the report.

The base file report has the effect of subtracting all statistics accumulated before the base file was created (e.g., this may be used to remove the time elapsing between system reset and 8:30 in the morning, when few if any users are on the system).

If a file name entered does not exist, BTMPM will requery for a new name.

Example:

```
   REPORT BTM STATISTICS ?
  ?Y(RET)
   REPORT STATUS SINCE RESET ?
  ?N(RET)
   USE BASE FILE ?
  ?Y(RET)
   ENTER FILE NAME BELOW.
→?XXYYZZ(RET)
   ARE HISTOGRAMS DESIRED?
  ?Y(RET)
 →ERROR OCCURRED.  FILE MAY NOT EXIST
   ENTER FILE NAME BELOW.
  ?
```

REPORT HISTORY FILE

If the user had not asked for a base file report, the history report query would follow.

Example:

```
   REPORT BTM STATISTICS ?
  ?Y(RET)
   REPORT STATUS SINCE RESET ?
  ?N(RET)
   USE BASE FILE?
  ?N(RET)
   USE HISTORY FILE?
→?Y(RET)
   ENTER FILE NAME BELOW.
  ?FILEHIST(RET)
   SUMMARY ONLY ?
  ?Y(RET)
   ARE HISTOGRAMS DESIRED?
  ?Y(RET)
```

The report covering the entire period of history file creation would then follow.

A "Y" answer to the history file question results in a query to enable the user to identify the history file to be processed. The next question allows the user to skip the snapshot listings and receive a summary only. An "N" answer to the summary only question will cause the snapshots and the summary to be listed. The histogram question follows the summary question, allowing the user to eliminate the time-consuming snapshot histogram printing. The terminal user might wish the summary only, to reduce the printing volume.

The snapshot statistics and histograms are more conveniently obtained by submitting a batch job.

If the file name is erroneous, BTMPM will detect the error.

Example:

```
   REPORT BTM STATISTICS ?
  ?Y(RET)
   REPORT STATUS SINCE RESET ?
  ?N(RET)
   USE BASE FILE ?
  ?N(RET)
   USE HISTORY FILE ?
  ?Y(RET)
   ENTER FILE NAME BELOW.
→?XXYYZZ (RET)
   SUMMARY ONLY ?
  ?Y(RET)
   ARE HISTOGRAMS DESIRED?
  ?Y(RET)
 →ERROR OCCURRED.  FILE MAY NOT EXIST
   ENTER FILE NAME BELOW.
  ?
```

When the history file summary has been completely printed the Report Generator will ask for more history files.

Example:

```
   MORE HISTORY FILES ?
  ?N(RET)
```

An "N" answer causes the chronological listing of important data from each snapshot in the history file. This listing will be followed by a sorted listing that groups snapshots with similar load characteristics.

A "Y" answer to the MORE HISTORY FILES? question allows the user to identify another history file and cycle through the SUMMARY ONLY? and ARE HISTOGRAMS DESIRED? questions. When the requested listings are completed, the Report Generator will ask for more history files. An "N" answer will terminate the history file report loop and initiate the chronological snapshot listing and sorted snapshot listing for all the history files processed.

BATCH PROCESSING OF HISTORY FILES

A batch job can be submitted to obtain complete history file listings with snapshot statistics. An example of a job set-up that will process the two history files, SEP04HIS and SEP08HIS is shown in Example 7.

A user may also ASSIGN F:108 (FILE, BTMLIST), and run the BTMPM Report Generator from the terminal by anticipating the questions that will now be written in the file BTMLIST. When the program is finished, file BTMLIST may be listed using FMGE in a batch job.

DEFINITION OF REPORTED STATISTICS

The format of the user statistics, task statistics, quanta statistics, performance statistics, and subsystems statistics are given in Figures 12 through 16 respectively.

Example 7. History File Processing

```
!JOB
!RUN (LMN,STAT)
!DATA
YES           (REPORT BTM STATISTICS ?)
NO            (REPORT STATUS SINCE RESET ?)
NO            (USE BASE FILE ?)
YES           (USE HISTORY FILE ?)
SEP04HIS      (ENTER FILE NAME BELOW ?)
NO            (SUMMARY ONLY ?)
YES           (ARE HISTOGRAMS DESIRED ?)
YES           (MORE HISTORY FILES ?)
SEP08HIS      (ENTER FILE NAME BELOW ?)
NO            (SUMMARY ONLY ?)
YES           (ARE HISTOGRAMS DESIRED ?)
NO            (MORE HISTORY FILES ?)
!EOD
```

```
TIME SAMPLE =             MINUTES
USERS LOGGED=
INTENSITY   =             COMPUTE MSEC/USER MIN
```

where

TIME SAMPLE is the duration of the sample period in minutes. If the report is for the status since reset, the sample time is measured from the time of system reset to the time the report is demanded. If a base file is used, the sample time is measured from the time of base file creation to the time the report was demanded. If the report is a snapshot, the sample time is the period between two consecutive snapshot records in the history file. If the report is a history file summary, the sample time is the period between the first and last snapshot records in the history file.

USERS LOGGED is the average number of on-line users. This statistic is tabulated each quantum and averaged over all quanta to account for users logging on and off during the time sample.

INTENSITY is obtained by multiplying the average interaction rate by the average task time. This statistic measures the amount of computing required per user minute. A "user minute" is defined as a minute of thinking and typing time. Output printing time and response time is not counted as user time.

Figure 12. User Statistics Format

```
                            INTER-   COMPUTE     TOTAL
                            ACTIVE     BOUND
NUMBER OF TASKS      =
% INPUT TASKS        =                                %
% SECONDARY TASKS    =                                %
AVERAGE TASK LENGTH  =                                MSEC
RESPONSE TIME        =                                MSEC
% ON-LINE TIME       =                                %
```

where

NUMBER OF TASKS is the total number of input tasks and secondary system tasks completed in the sample. If a task required only one quantum, the task is considered interactive and is tabulated in the first column. If more than one quantum was required, the task is considered compute bound and is tabulated in the second column. The total number of tasks is shown in the third column.

% INPUT TASKS is the percentage of the total number of tasks initiated by an input activation character. At the executive level, all characters are activation characters (i.e., A and S for the ASSIGN command). At the subsystem and user level, the activation characters are usually carriage returns. See Appendix C for a complete definition of activation characters.

% SECONDARY TASKS is the percentage of the total number of tasks generated by the system as a result of output bound tasks and those which result because a user depresses two consecutive escape characters. A task is dismissed as output bound if a Teletype output buffer is filled (100 characters). The dismissed task will be queued for service when less than 10 characters remain to be printed (one second lead time for Teletypes). Two escape characters will cause the user to be queued along with other output bound users. If the resulting secondary task requires more than one quantum, the task is tabulated as a compute bound secondary task. Usually, an output bound task needs only enough time to fill the 100 character buffer; this time typically results in a very short interactive task.

Figure 13. Task Statistics Format

AVERAGE TASK LENGTH is the average task time in milliseconds, averaged separately for interactive and compute bound tasks. The third column is the average task length for all tasks. The task time includes CPU time and file I/O time. The task must complete within the sample to be included in this statistic.

RESPONSE TIME is the time measured in milliseconds from the receipt of an activation character to the end of the last quantum of the task. The end of the last quantum corresponds closely to the insertion of the Teletype prompt character in the output buffer. Activation characters "typed ahead" are processed along with the other input characters until the input buffer is empty. The first column contains the interactive response, which is the time required to receive one quantum of service. Column 2 contains the compute bound response which varies, depending upon the length of the compute bound tasks that terminate in the sample. Since compute bound task lengths vary widely, the ratio of response time to task time is more meaningful. This ratio is computed below and is labeled the execution time multiplication factor, ETMF.

% ON-LINE TIME is the percentage of the sample time used for on-line processing. Interactive percent on-line time is 100 times the total interactive task time divided by the sample time. The compute bound percent on-line time is 100 times (the total on-line time minus the interactive task time) divided by the sample time. This statistic is accurate at the end of each quantum even though some compute bound tasks are not completed in the sample time. The total percent used on-line is dependent upon the choice of the three preset quanta and upon the intensity of the work load.

Figure 13. Task Statistics Format (cont.)

```
                         LEVEL 1    LEVEL 2
NUMBER OF QUANTA     =
MAXIMUM QUANTUM      =                      MSEC
PRESET QUANTUM       =                      MSEC
AVERAGE QUANTUM      =                      MSEC
% EXTENDED QUANTA    =                      %
% LONG EXT QUANTA    =                      %
AVERAGE EXTENSION    =                      MSEC
  STANDARD DEVIATION =
RAD  ACCESSES/EXT QTM=
DISK ACCESSES/EXT QTM=

SUBSYSTEM / MAX EXT  =
RAD  ACCESSES/MAX EXT=
DISK ACCESSES/MAX EXT=
PERCENT QUEUED USERS =
```

where

NUMBER OF QUANTA is the count of quanta for each level. All interactive tasks contain a single first level quantum. Compute bound tasks contain a single first level quantum and one or more second level quanta.

MAXIMUM QUANTUM is the longest quantum that occurred for each level since the system was started. This number increases throughout the day. The size of the maximum quantum is dependent upon the number of RAD and disk accesses and upon the length of time an operator has the system in an "idle" condition.

PRESET QUANTUM is the quantum length set at SYSGEN, key-in or system patch. For each of the two levels, the preset quantum length is the maximum time allowed for a respective user before dismissal. When a user fails to exhaust his demands, he is unconditionally queued for another quantum. File I/O unconditionally extends the quantum past the preset value for as long as needed to complete the operation. The value of q_1, the first level quantum preset, determines the percentage of tasks that will be completed in one quantum. The value of q_2, the second level quantum present determines the ETMF for compute bound tasks.

Figure 14. Quanta Statistics Format

AVERAGE QUANTUM is computed by accumulating quantum lengths for each level and dividing by the number of quanta in each level. The average first level quantum includes the first quantum of each compute bound task and the quanta from all interactive tasks. The average second level quantum includes all quanta in compute bound tasks with the exception of the first quantum. The self-dismissed second level quanta are not tabulated in the average second level quantum since they would bias the average.[†]

% EXTENDED QUANTA is the percent of the quanta for each level extended for file I/O completion or for the case in which the idle switch is in an IDLE state during an on-line quantum. File I/O extension is an important factor in BTM performance since extensions incur longer response times for all users queued at the time of extension.

% LONG EXT QUANTA is the percentage of quanta for each level extended more than one first level preset quantum. This percentage is included in % EXTENDED QUANTA.

AVERAGE EXTENSION is the total time that quanta were extended past the preset quantum value divided by the number of extensions. It may be observed that "write-check" will incur longer extensions since two accesses are required for all file write operations.

STANDARD DEVIATION is the square root of the expected squared extension minus the average extension squared. This statistic is a measure of dispersion in the distribution for quantum extensions.

RAD ACCESSES/EXT QTM is the average number of RAD accesses experienced during extended quanta. This statistic is useful since quantum extensions are primarily due to mutiple access file operations such as the M:CLOSE.

DISK ACCESSES/EXT QTM is the average number of disk pack accesses experienced during extended quanta. This statistic is important since disk access times are much greater than the RAD accesses, and have a large effect on system operation.

SUBSYSTEM/MAX EXT identifies the subsystem that was executing when the maximum quantum extension occurred. The subsystem is identified for both quantum levels.

RAD ACCESSES/MAX EXT is the number of RAD accesses that occurred during the maximum quantum extension for a given quantum level.

DISK ACCESSES/MAX EXT is the number of disk pack accesses that occurred during the maximum quantum extension for a given quantum level.

PERCENT QUEUED USERS is the average percentage of the logged-on users that were queued for a given quantum level. This statistic is accumulated at the end of each on-line quantum and averaged over the number of quanta.

[†] A quantum is considered "self-dismissed" if a CAL1,8 FPT X'40' is executed. Execution of this CAL causes the current quantum to be terminated. The associated user is then queued for service in the compute bound queue.

Figure 14. Quanta Statistics Format (cont.)

```
TASK COMPLETION RATE =         TASKS/LOGGED MINUTE
INTERACTION RATE     =         TASKS/USER MINUTE
ETMF FOR COMPUTE BND =         USER TIME / COMPUTE TIME

CORE PAGES PROCEDURE =
CORE PAGES DATA      =
ON-LINE RAD ACCESSES =
ON-LINE DISK ACCESSES=
ON-LINE RAD  ACCESSES/INPUT REQUEST =
ON-LINE DISK ACCESSES/INPUT REQUEST =

SHORT LEVEL 2 QUANTA =
SHORT HISTORY QUANTA =
AVERAGE BATCH QUANTUM=
ACTIVATION CHARACTERS=
```

where

TASK COMPLETION RATE is the total number of tasks in the sample divided by the total log-on time for all users in the sample. This is a measure of on-line task throughput.

INTERACTION RATE is a measure of user activity that does not depend upon the response of the system. This statistic is obtained by dividing the total number of requests by the average number of users that are not queued or being serviced times the sample time. In effect, this represents the rate at which a user "thinks and types".

ETMF FOR COMPUTE BND The execution time multiplication factor, ETMF, is the average ratio of compute bound response to compute bound task time. The ETMF is computed only for input tasks that require two or more quanta. This statistic is one of the most useful measures of system performance. A sharply rising value of ETMF will indicate the saturation of the system. Interactive response will not reflect the saturation point because of preferential scheduling.

CORE PAGES PROCEDURE is the average number of pages allocated to type (01) control sections. This includes all instruction areas that would normally be write protected. The histogram for pages of pure procedure may be included in the report, and is a descriptive measure of core usage. This data day be useful when considering future systems that utilize paging. "CORE PAGES PROCEDURE" is an average number computed over all on-line quanta in the sample. The executive level of execution contains zero pages of pure procedure.

CORE PAGES DATA is the average number of pages allocated to type (00) storage, common, and dynamic common. The histogram for pages of data is a more descriptive measure of core usage. These pages are four data pages of context for each user. The executive level of execution requires exactly four data pages. This statistic is computed over all quanta in the sample.

ON-LINE RAD ACCESSES is the number of accesses used to support file I/O activity. Several RAD accesses may be required at high speed for each low-speed disk access since dictionaries and file control are maintained on the high-speed RAD.

ON-LINE DISK ACCESSES is the number of accesses that were used for on-line file I/O activity. This total is obtained directly from the BTM executive tables.

ON-LINE RAD ACCESSES/INPUT REQUEST is the average number of accesses per activation character. This number may include file I/O data accessing if disk packs are not used.

ON-LINE DISK ACCESSES/INPUT REQUEST is the average number of accesses per activation character. This is only one of the possible averages that could be computed. Other averages, such as accesses per minute, can be done by hand. Disk pack accesses can contribute substantially to quantum extensions due to multiple accesses in lengthy file operations.

Figure 15. Performance Statistics Format

SHORT LEVEL 2 QUANTA   is a count of all second level quanta that required two milliseconds or less, and were not the last quantum of a task. The count indicates the number of self-dismissing quanta in the sample. This count includes the BTMPM history file "sleeping quanta" as well as "sleeping quanta" from other sources.

SHORT HISTORY QUANTA   is a count of quantum dismissals by the history file portion of BTMPM. This statistic is printed only for history file snapshots and history file summaries. The load placed on the system by the history file creation is computed by assuming that batch received one full quantum for each sleeping quantum, thereby reducing the percentage of the system available to on-line.

AVERAGE BATCH QUANTUM   is computed only for history files by assigning all remaining time in the sample to batch, and dividing by the total number of on-line quanta. The history file creation program is always queued, which means that the number of batch quanta is the same as the number of on-line quanta.

ACTIVATION CHARACTERS   is the count of input activation characters that initiated tasks. Activation characters that are typed ahead do not initiate tasks individually, since the first activation character starts a task that will continue until the input buffer is empty.

Figure 15. Performance Statistics Format (cont.)

```
             ON-LINE              MEAN     STANDARD
              TIME      TASKS     TASK    DEVIATION
              (%)        (%)     (MSEC)    (MSEC)
   BASIC      37         38       39        40
   BPM
   DELTA
   EDIT
   FERRET
   FORTRAN
   LOAD
   MANAGE
   RUN
   SUPER
   SYMBOL
   EXECUTIVE
```

where

ON-LINE TIME   for each subsystem is obtained by dividing the sum of the quanta used per subsystem by the total amount of on-line time and converting to percent. The profile of the user environment will be indicated by this column of percentages which reflect subsystem utilization.

TASKS   for each subsystem is the number of tasks completed in the subsystem divided by the total number of on-line tasks.

MEAN TASK   is the average task length for each subsystem.

STANDARD DEVIATION   is the square root of the expected squared task time minus the average task squared. This is a measure of dispersion for the task time distribution for each subsystem.

Figure 16. Subsystem Statistics Format

HISTOGRAMS

There are four different types of histograms available to the BTMPM systems management user, as follows:

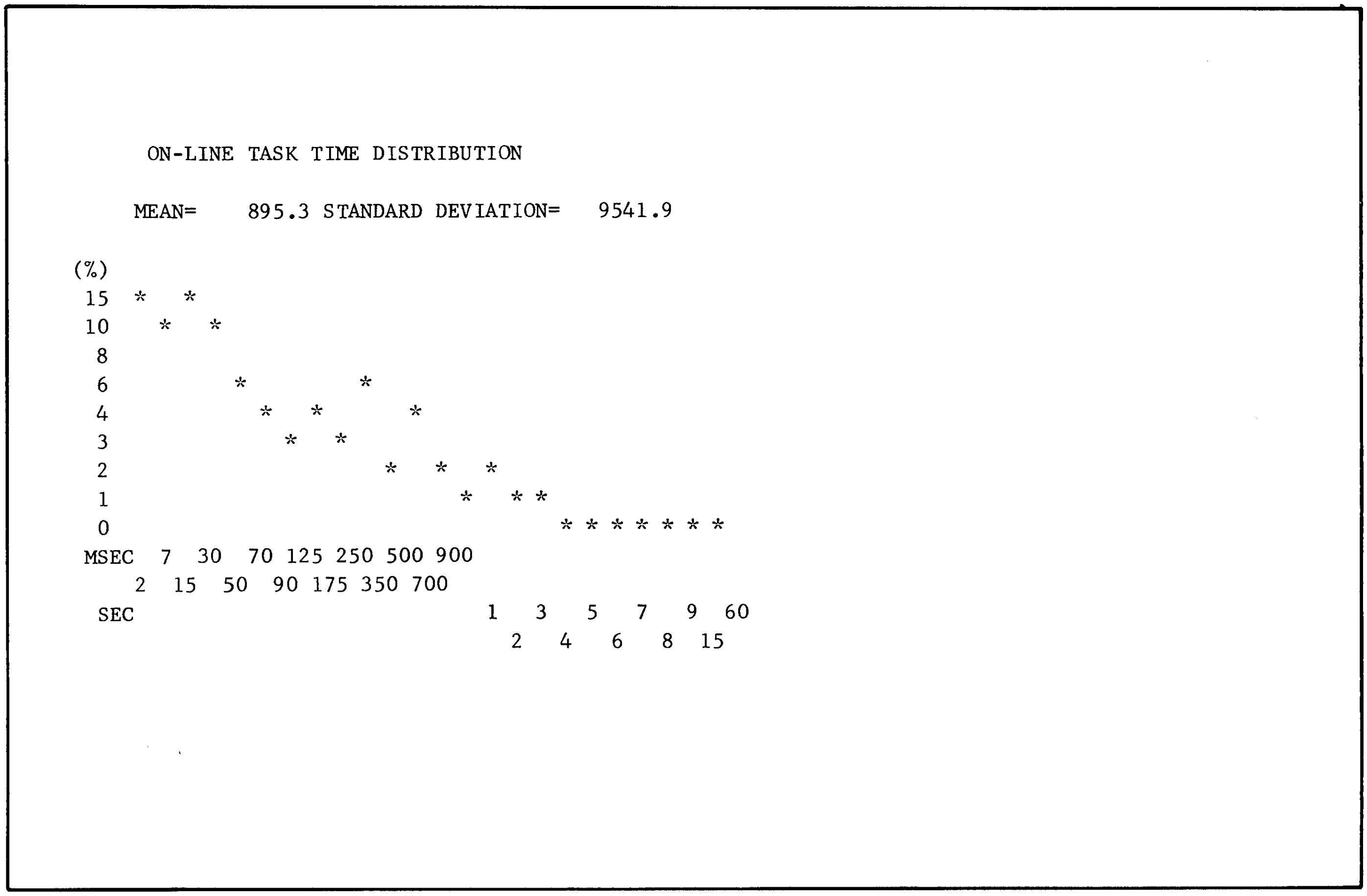
1. ON-LINE TASK TIME DISTRIBUTION is a histogram showing the frequency of occurrence for all tasks. This includes interactive input tasks, compute bound input tasks and output bound system tasks. An entry in a given percentage indicates that there were enough requests to exceed the number on the percentage scale at the left. For instance, a point with 0 percent indicates at least one request but less than 1 percent of all the requests corresponded to a particular value on the horizontal axis. The values shown on the horizontal axis represent the centers of the discrete windows that were used to accumulate the data. The mean and variance are computed from the histogram. Note that the scale is nonlinear. This histogram is useful when choosing a value of the first level preset quantum, q_1, which will ensure that 80 to 90 percent of all tasks will complete in one quantum. The format and sample data for this histogram are shown in Example 8.

2. INTERACTIVE RESPONSE TIME DISTRIBUTION is a histogram showing the frequency of occurrence of response times for tasks that took less than one preset quantum. In general, all tasks on the CPU task time distribution that are to the left of one preset quantum length are tabulated on this response time chart. The scale for this plot is also nonlinear. The format and sample data for this histogram are shown in Example 9.

3. PURE PROCEDURE DISTRIBUTION is a histogram showing the frequency of occurrence for memory pages of pure procedure (i.e., the amount of pure procedure required in the servicing of on-line requests). This plot will have a high count at zero pages since the executive level has no pages of pure procedure. Other high points may vary with subsystems written to use fixed amounts of pure procedure. The format and sample data for this histogram are shown in Example 10.

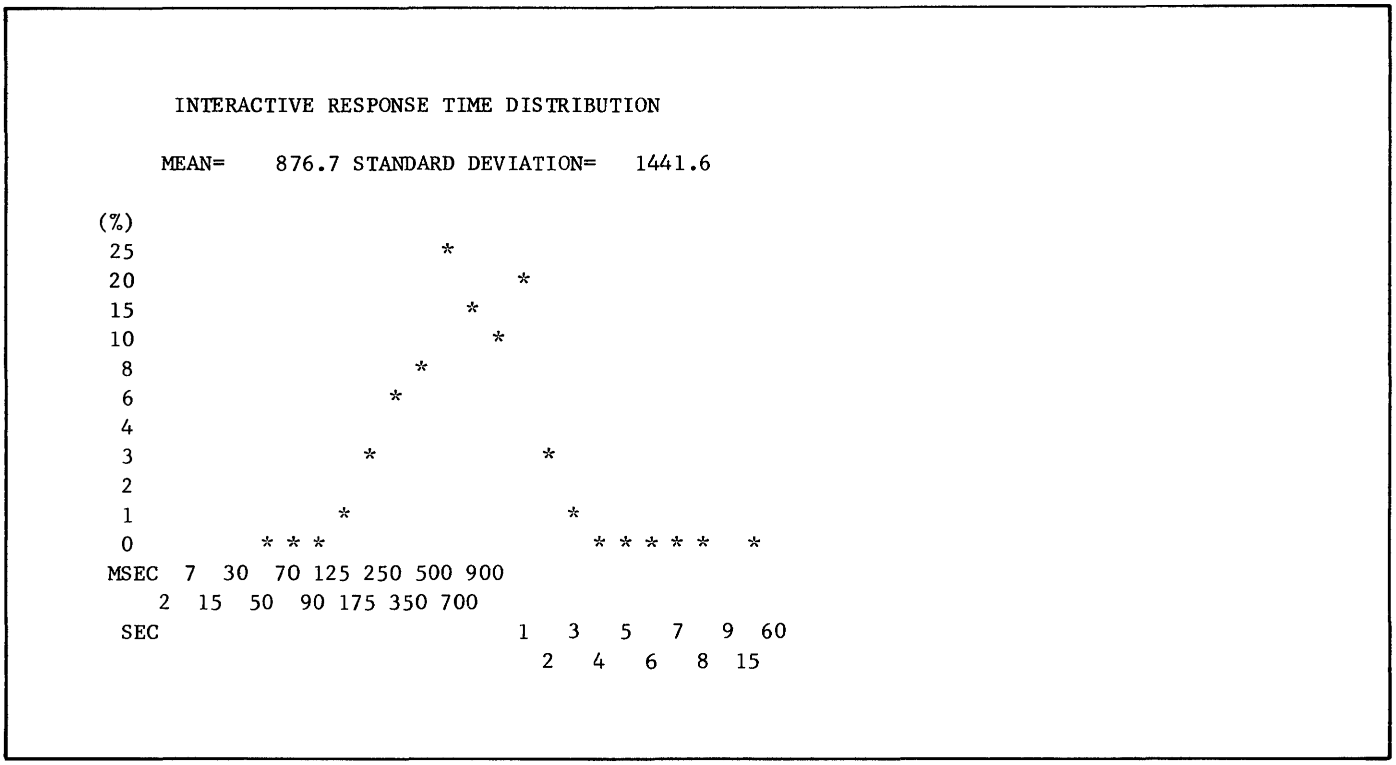
4. DATA DISTRIBUTION is a histogram showing the frequency of occurrence for data pages. This includes common and dynamic common pages. There is a high point at four pages, since the executive level requires only the four pages of context area. The format and sample data for this histogram are shown in Example 11.

Example 8. On-Line Task Time Distribution



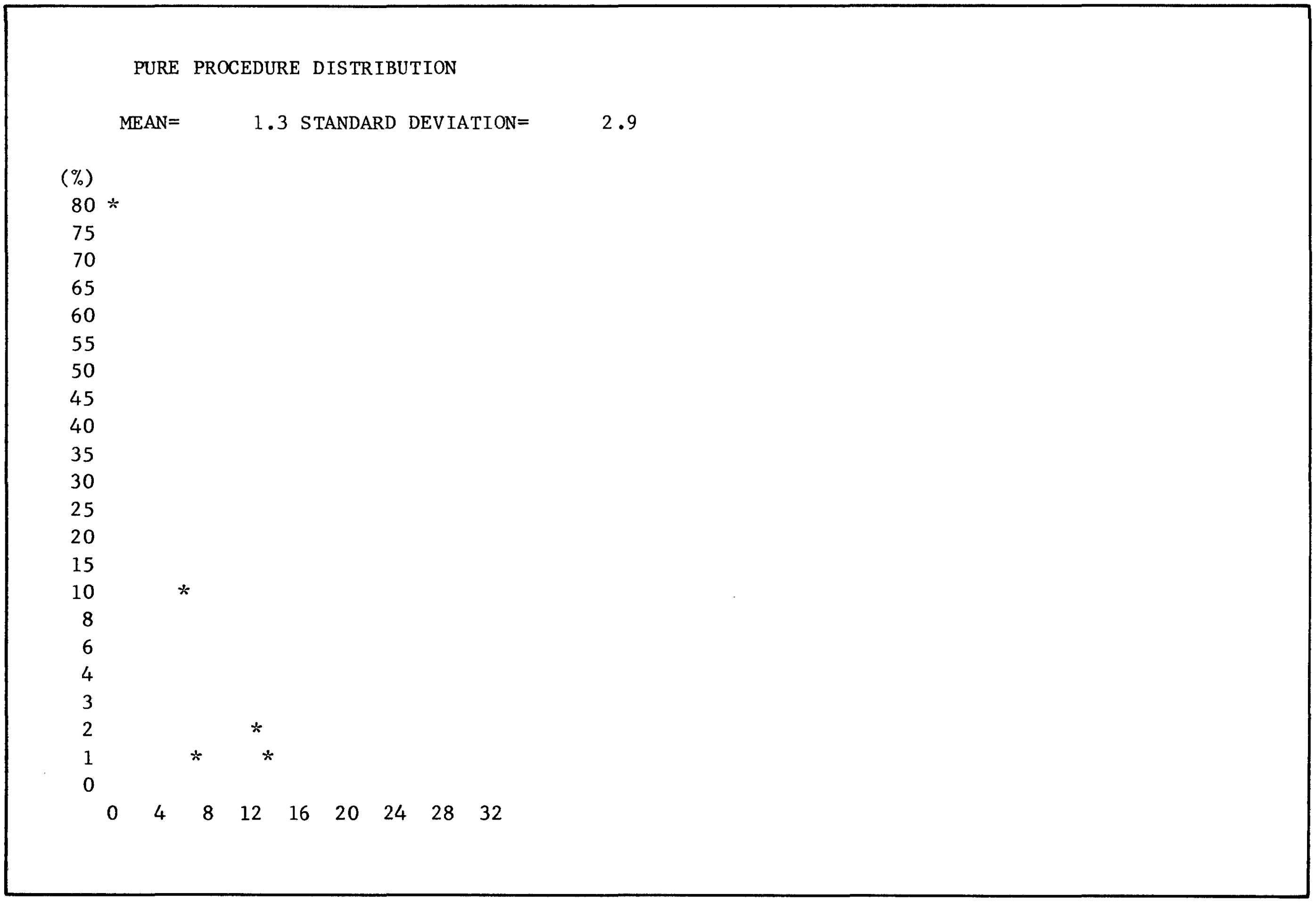
```
      ON-LINE TASK TIME DISTRIBUTION

     MEAN=    895.3 STANDARD DEVIATION=   9541.9

(%)
 15  *   *
 10    *   *
  8
  6          *         *
  4            *   *       *
  3              *   *
  2                      *   *   *
  1                            *   * *
  0                                    * * * * * * *
 MSEC  7  30  70 125 250 500 900
     2  15  50  90 175 350 700
  SEC                           1   3   5   7   9  60
                                  2   4   6   8  15
```

Example 9. Interactive Response Time Distribution



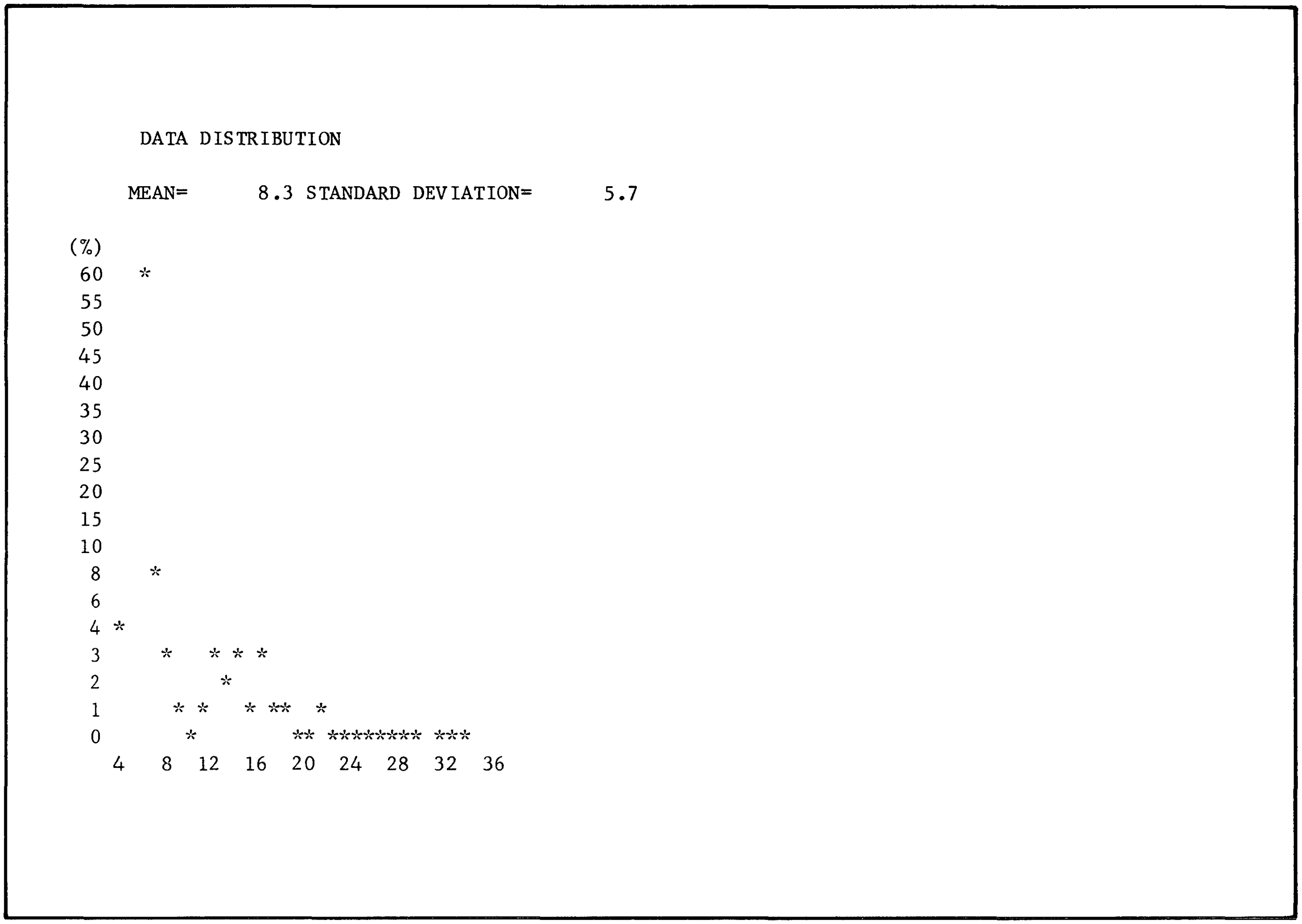
```
      INTERACTIVE RESPONSE TIME DISTRIBUTION

     MEAN=    876.7 STANDARD DEVIATION=   1441.6

(%)
 25                        *
 20                              *
 15                          *
 10                            *
  8                      *
  6                    *
  4
  3                  *             *
  2
  1                *                 *
  0          * * *                     * * * * *   *
 MSEC  7  30  70 125 250 500 900
     2  15  50  90 175 350 700
  SEC                            1   3   5   7   9  60
                                   2   4   6   8  15
```

Example 10. Pure Procedure Distribution



```
      PURE PROCEDURE DISTRIBUTION

     MEAN=      1.3 STANDARD DEVIATION=      2.9

(%)
 80 *
 75
 70
 65
 60
 55
 50
 45
 40
 35
 30
 25
 20
 15
 10       *
  8
  6
  4
  3
  2             *
  1        *     *
  0
    0   4   8  12  16  20  24  28  32
```

Example 11. Data Distribution



```
      DATA DISTRIBUTION

     MEAN=      8.3 STANDARD DEVIATION=      5.7

(%)
 60   *
 55
 50
 45
 40
 35
 30
 25
 20
 15
 10
  8    *
  6
  4 *
  3     *   * * *
  2          *
  1      * *   * **  *
  0       *        ** ******** ***
    4   8  12  16  20  24  28  32  36
```

SNAPSHOT SUMMARY

The snapshot summary format is given in Figure 17.

SNAPSHOT TIME SUMMARY

The important statistics obtained from each snapshot are printed in chronological order under the headings given in Figure 17. The time of the end of the snapshot and the date will appear at the right of each snapshot entry. If more than one history file was processed, the files will appear in the order in which they were processed.

SORTED SNAPSHOT SUMMARY

The above chronological list of snapshot summaries is sorted by the number of users and by intensity. The intensity ranges in intervals of 200 millisecond/user minute from zero to 3000 millisecond/user minute. The snapshots listed must have an average batch quantum within 15 percent of the average for the sample. The snapshots must also have a mix of compute bound time to interactive time that is within 15 percent of the average mix. These restrictions allow one to compare snapshots of similar user loads and ignore snapshots that represent extreme cases, while allowing the user to see all cases in the time summary.

SAMPLE HISTORY SUMMARY

The history file report given in Figures 18 through 20 is an example of the BTMPM report described in preceding sections. The history consists of 28 records, 20 minutes apart, beginning at 08.24 in the morning and ending at 17.24 in the afternoon. The report includes the History File Summary (without Histograms), the Snapshot Summary, and the Sorted Snapshot Summary.

The history file example was created by running the history file creation portion of BTMPM for the entire nine-hour period. The report was obtained by processing the history file at a later date. The complete listing for each snapshot is not included in the example due to the volume of data.

```
AVERAGE INTENSITY    =          COMPUTE MSEC/USER MINUTE
AVERAGE TASK         =          MSEC
AVERAGE BATCH QUANTUM=          MSEC
ON-LINE TIME MIX     =          COMPUTE BOUND/INTERACTIVE
BTM SNAPSHOT SUMMARIES
```

where

AVERAGE INTENSITY is the average intensity for all of snapshots in the history files processed. The intensity is the product of the interaction rate and the average task time in each snapshot.

AVERAGE TASK is the average of the average task time for all the snapshots being processed.

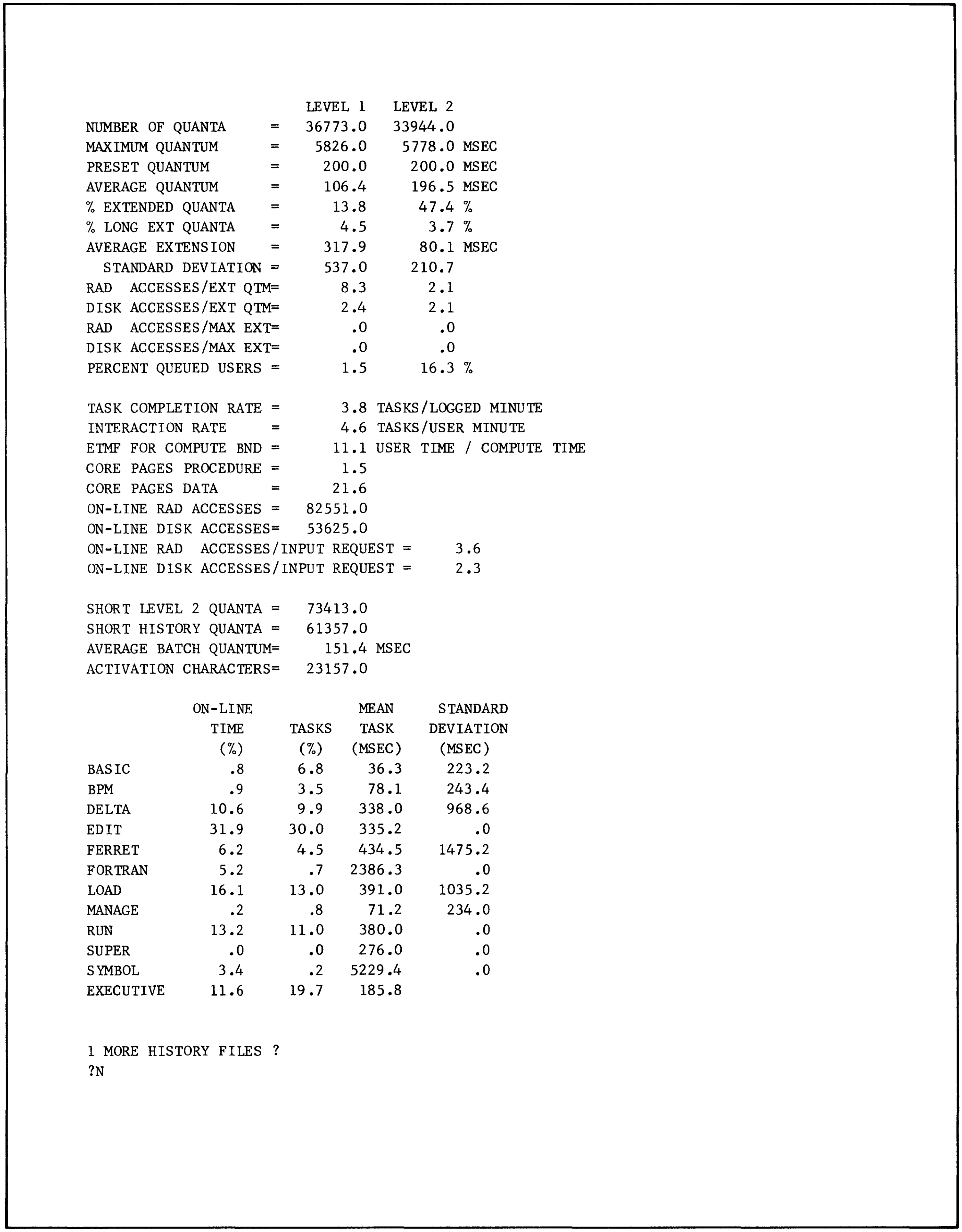
AVERAGE BATCH QUANTUM is the average of the average batch quantum for all the snapshots being processed.

ON-LINE TIME MIX is the average ratio of compute bound time to interactive time over all the snapshots in the sample. This statistic is useful to determine whether or not a given snapshot is heavily compute bound.

Figure 17. Snapshot Summary Format

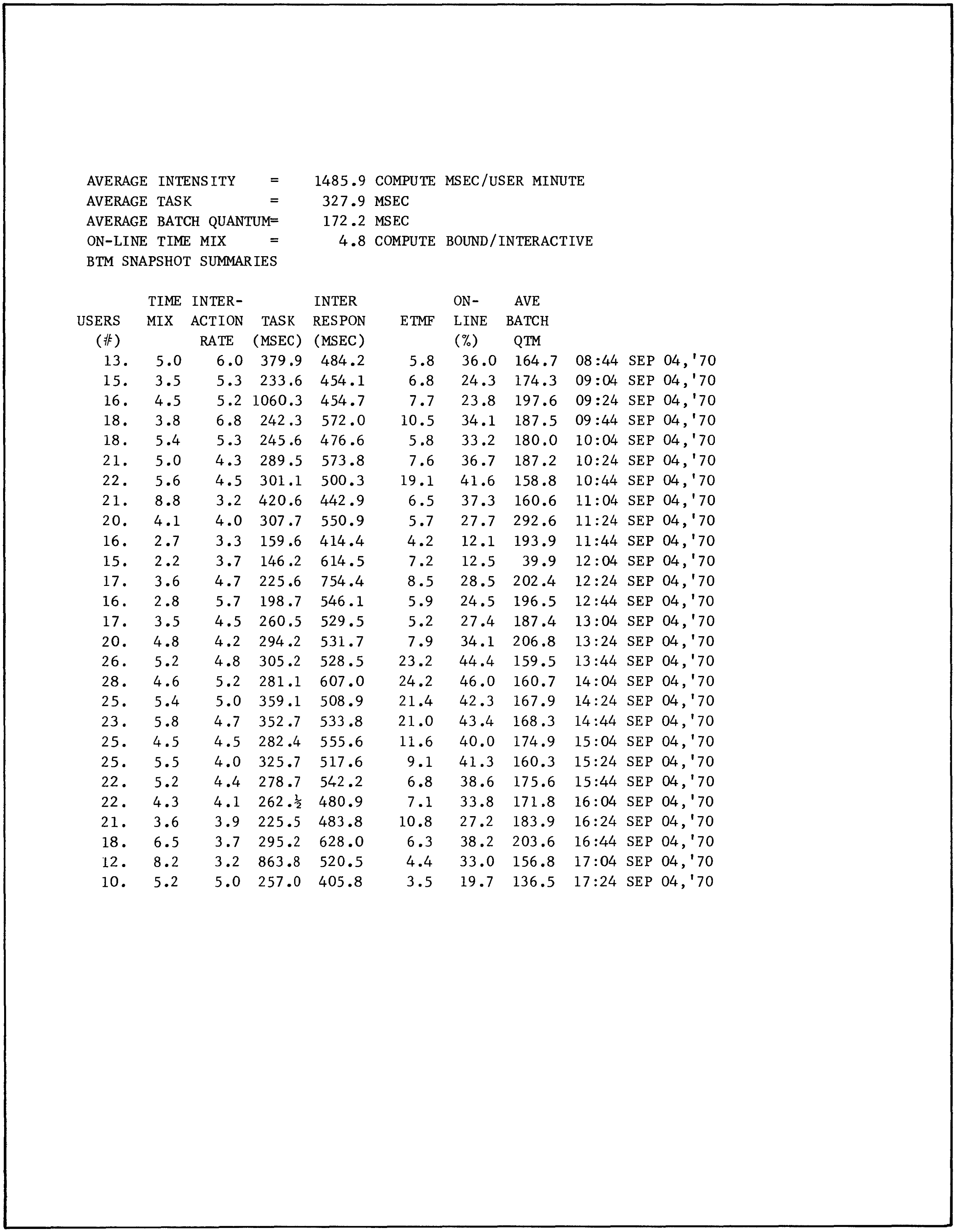
```
!RUN
LOAD MODULE FID:STAT
;G
*BTM PERFORMANCE MONITOR*
 REPLY WITH Y OR N OR X AND CARRIAGE RETURN.

 REPORT BTM STATISTICS ?
?Y
 REPORT STATUS SINCE RESET ?
?N
 USE BASE FILE ?
?N
 USE HISTORY FILE ?
?Y
 ENTER FILE NAME BELOW.
?SEP04HIS
 SUMMARY ONLY ?
?Y
 ARE HISTOGRAMS DESIRED?
?N

BTM HISTORY SUMMARY, FILE = SEP04HIS FROM 08:24 TO 17:24 SEP 04,'70
TIME SAMPLE =     540.0 MINUTES
USERS LOGGED=      18.4
INTENSITY   =    1435.9 COMPUTE MSEC/USER MIN
                           INTER-   COMPUTE     TOTAL
                           ACTIVE    BOUND
NUMBER OF TASKS      =   29884.0    6909.0   36793.0
% INPUT TASKS        =      51.6      11.3      62.9 %
% SECONDARY TASKS    =      29.6       7.5      37.1 %
AVERAGE TASK LENGTH  =      61.9    1412.1     315.5 MSEC
RESPONSE TIME        =     528.6   14296.7           MSEC
% ON-LINE TIME       =       5.7      26.9      32.7 %
```

Figure 18. History Summary Example



```
                         LEVEL 1   LEVEL 2
NUMBER OF QUANTA     =   36773.0   33944.0
MAXIMUM QUANTUM      =    5826.0    5778.0 MSEC
PRESET QUANTUM       =     200.0     200.0 MSEC
AVERAGE QUANTUM      =     106.4     196.5 MSEC
% EXTENDED QUANTA    =      13.8      47.4 %
% LONG EXT QUANTA    =       4.5       3.7 %
AVERAGE EXTENSION    =     317.9      80.1 MSEC
  STANDARD DEVIATION =     537.0     210.7
RAD  ACCESSES/EXT QTM=       8.3       2.1
DISK ACCESSES/EXT QTM=       2.4       2.1
RAD  ACCESSES/MAX EXT=        .0        .0
DISK ACCESSES/MAX EXT=        .0        .0
PERCENT QUEUED USERS =       1.5      16.3 %

TASK COMPLETION RATE =       3.8 TASKS/LOGGED MINUTE
INTERACTION RATE     =       4.6 TASKS/USER MINUTE
ETMF FOR COMPUTE BND =      11.1 USER TIME / COMPUTE TIME
CORE PAGES PROCEDURE =       1.5
CORE PAGES DATA      =      21.6
ON-LINE RAD ACCESSES =   82551.0
ON-LINE DISK ACCESSES=   53625.0
ON-LINE RAD  ACCESSES/INPUT REQUEST =     3.6
ON-LINE DISK ACCESSES/INPUT REQUEST =     2.3

SHORT LEVEL 2 QUANTA =   73413.0
SHORT HISTORY QUANTA =   61357.0
AVERAGE BATCH QUANTUM=     151.4 MSEC
ACTIVATION CHARACTERS=   23157.0

            ON-LINE             MEAN    STANDARD
              TIME     TASKS    TASK    DEVIATION
              (%)       (%)    (MSEC)     (MSEC)
BASIC          .8       6.8      36.3     223.2
BPM            .9       3.5      78.1     243.4
DELTA        10.6       9.9     338.0     968.6
EDIT         31.9      30.0     335.2        .0
FERRET        6.2       4.5     434.5    1475.2
FORTRAN       5.2        .7    2386.3        .0
LOAD         16.1      13.0     391.0    1035.2
MANAGE         .2        .8      71.2     234.0
RUN          13.2      11.0     380.0        .0
SUPER          .0        .0     276.0        .0
SYMBOL        3.4        .2    5229.4        .0
EXECUTIVE    11.6      19.7     185.8


1 MORE HISTORY FILES ?
?N
```

Figure 18. History Summary Example (cont.)



```
 AVERAGE INTENSITY    =     1485.9 COMPUTE MSEC/USER MINUTE
 AVERAGE TASK         =      327.9 MSEC
 AVERAGE BATCH QUANTUM=      172.2 MSEC
 ON-LINE TIME MIX     =        4.8 COMPUTE BOUND/INTERACTIVE
 BTM SNAPSHOT SUMMARIES

        TIME INTER-        INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE  BATCH
 (#)          RATE  (MSEC) (MSEC)         (%)    QTM
  13.   5.0    6.0  379.9  484.2     5.8  36.0  164.7  08:44 SEP 04,'70
  15.   3.5    5.3  233.6  454.1     6.8  24.3  174.3  09:04 SEP 04,'70
  16.   4.5    5.2 1060.3  454.7     7.7  23.8  197.6  09:24 SEP 04,'70
  18.   3.8    6.8  242.3  572.0    10.5  34.1  187.5  09:44 SEP 04,'70
  18.   5.4    5.3  245.6  476.6     5.8  33.2  180.0  10:04 SEP 04,'70
  21.   5.0    4.3  289.5  573.8     7.6  36.7  187.2  10:24 SEP 04,'70
  22.   5.6    4.5  301.1  500.3    19.1  41.6  158.8  10:44 SEP 04,'70
  21.   8.8    3.2  420.6  442.9     6.5  37.3  160.6  11:04 SEP 04,'70
  20.   4.1    4.0  307.7  550.9     5.7  27.7  292.6  11:24 SEP 04,'70
  16.   2.7    3.3  159.6  414.4     4.2  12.1  193.9  11:44 SEP 04,'70
  15.   2.2    3.7  146.2  614.5     7.2  12.5   39.9  12:04 SEP 04,'70
  17.   3.6    4.7  225.6  754.4     8.5  28.5  202.4  12:24 SEP 04,'70
  16.   2.8    5.7  198.7  546.1     5.9  24.5  196.5  12:44 SEP 04,'70
  17.   3.5    4.5  260.5  529.5     5.2  27.4  187.4  13:04 SEP 04,'70
  20.   4.8    4.2  294.2  531.7     7.9  34.1  206.8  13:24 SEP 04,'70
  26.   5.2    4.8  305.2  528.5    23.2  44.4  159.5  13:44 SEP 04,'70
  28.   4.6    5.2  281.1  607.0    24.2  46.0  160.7  14:04 SEP 04,'70
  25.   5.4    5.0  359.1  508.9    21.4  42.3  167.9  14:24 SEP 04,'70
  23.   5.8    4.7  352.7  533.8    21.0  43.4  168.3  14:44 SEP 04,'70
  25.   4.5    4.5  282.4  555.6    11.6  40.0  174.9  15:04 SEP 04,'70
  25.   5.5    4.0  325.7  517.6     9.1  41.3  160.3  15:24 SEP 04,'70
  22.   5.2    4.4  278.7  542.2     6.8  38.6  175.6  15:44 SEP 04,'70
  22.   4.3    4.1  262.½  480.9     7.1  33.8  171.8  16:04 SEP 04,'70
  21.   3.6    3.9  225.5  483.8    10.8  27.2  183.9  16:24 SEP 04,'70
  18.   6.5    3.7  295.2  628.0     6.3  38.2  203.6  16:44 SEP 04,'70
  12.   8.2    3.2  863.8  520.5     4.4  33.0  156.8  17:04 SEP 04,'70
  10.   5.2    5.0  257.0  405.8     3.5  19.7  136.5  17:24 SEP 04,'70
```

Figure 19. Chronological Snapshot Summary Example

```
  INTENSITY =   400. TO   600. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  16.    2.7    3.3  159.6  414.4    4.2   12.1  193.9  10:24 SEP 04,'70

  INTENSITY =   800. TO  1000. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  21.    3.6    3.9  225.5  483.8   10.8   27.2  183.9  13:44 SEP 04,'70

  INTENSITY =  1000. TO  1200. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  17.    3.5    4.5  260.5  529.5    5.2   27.4  187.4  11:44 SEP 04,'70

  INTENSITY =  1200. TO  1400. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  15.    3.5    5.3  233.6  454.1    6.8   24.3  174.3  10:04 SEP 04,'70

  18.    5.4    5.3  245.6  476.6    5.8   33.2  180.0  12:24 SEP 04,'70

  20.    4.1    4.0  307.7  550.9    5.7   27.7  191.6  13:04 SEP 04,'70

  21.    5.0    4.3  289.5  573.8    7.6   36.7  187.2  14:04 SEP 04,'70
  21.    8.8    3.2  420.6  442.9    6.5   37.3  160.6  14:24 SEP 04,'70

  22.    5.2    4.4  278.7  542.2    6.8   38.6  175.6  15:04 SEP 04,'70
  22.    5.6    4.5  301.1  500.3   19.1   41.6  158.8  15:24 SEP 04,'70

  25.    4.5    4.5  282.4  555.6   11.6   40.0  174.9  16:04 SEP 04,'70
  25.    5.5    4.0  325.7  517.6    9.1   41.3  160.3  16:24 SEP 04,'70

  INTENSITY ,  1400. TO  1600. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  28.    4.6    5.2  281.1  607.0   24.2   46.0  160.7  17:24 SEP 04,'70

  INTENSITY =  1600. TO  1800. COMPUTE MSEC / USER MINUTE

        TIME INTER-         INTER          ON-    AVE
USERS   MIX  ACTION  TASK  RESPON   ETMF  LINE   BATCH
 (#)          RATE  (MSEC) (MSEC)          (%)    QTM
  18.    3.8    6.8  242.3  571.0   10.5   34.1  187.5  12:44 SEP 04,'70

  23.    5.8    4.7  352.7  533.8   21.0   43.4  168.3  15:44 SEP 04,'70
```

Figure 20. Sorted Snapshot Summary Examples

SYSTEM TUNING PROCEDURES

There are three optional tuning parameters that may be used by the system manager to control system performance. The function of each parameter and rationale for the recommended starting points is as follows:

1. q_B is the size of the batch quantum. This parameter may be set at SYSGEN and changed dynamically via an operator key-in. The installation manager need only control q_B to tune the system. For a primarily on-line system, the batch quantum should be low enough to ensure swap limited operation. For batch systems, the value of q_B may be increased from the starting point when the number of users drops into the next lower range as shown in Tables 1 and 2 (see below). This guarantees the batch job stream larger throughput with lower user loads.

2. q_1 is the size of the first level on-line quantum. This may be set at SYSGEN and be changed dynamically via an operator key-in. the choice of q_1 = 200 msec ensures that 80 percent to 85 percent of all on-line tasks will be completed in one quantum. The choice of q_1 = 300 msec assures that 85 percent to 90 percent of the tasks will complete in one quantum (an increase of q_1 beyond 300 msec is not justified). The value of q_1 determines how many tasks may be completed in one quantum.

3. q_2 is the size of the second level on-line quantum. This may be set at SYSGEN and be changed dynamically via an operator key-in. Once q_1 is chosen, the value of the multiplier must be chosen to ensure on-line users an adequate percentage of system time as determined by the formula given above for minimum batch percentage. The value of q_2 has a small effect on interactive response since only compute bound tasks receive q_2 quanta. The choice of a higher multiplier will result in lower batch throughput.

It is important to recognize that, typically, each on-line user requires from 1 percent to 3 percent of the system time. The quanta may be adjusted to one of the following criteria:

1. Increase on-line service to move the saturation point higher.

2. Decrease on-line service to provide a higher batch throughput.

TUNING IMPLEMENTATION

A procedure to perform system tuning is as follows:

1. Choose a set of starting values for q_B, q_1, and q_2 from Table 1 or 2, depending upon the swapping device used. Choose the user range that includes the heaviest anticipated user load.

Table 1. Recommended Starting Points Using 7212[t]

User Range	Minimum Batch	Starting Values
		$q_{batch}/q_1/q_2$
0-10	66%	400/200/200
10-20	50%	200/200/200
20-40	21%	q_B/200/400
40-60	12%	q_B/200/800

[t] Average swap cycle = 110 msec.
q_B = 50 msec, swap limited case.

Table 2. Recommended Starting Points Using 7232[t]

User Range	Minimum Batch	Starting Values
		$q_{batch}/q_1/q_2$
0-10	57%	400/300/300
10-20	47%	q_B/300/300
20-50	18%	q_B/300/1200

[t] Average swap cycle = 273 msec.

2. Run the system and observe performance. At medium user loads, the interactive response should not change noticeably as more users log on because of preferential scheduling. The compute bound response will provide the best measure of response since system saturation will cause the ETMF to increase sharply.

3. For user loads that are lower than the range used for the starting point, the batch quantum may be

increased. This will guarantee a higher batch throughput, as determined by the formula

$$\text{minimum \% batch} = \frac{q_B}{q_B + q_2} \text{ where } q_2 \geq q_1$$

BTMPM INSTALLATION

The resident portion of BTMPM requires a 102-word table and other tables whose length is determined by the number of users (NUMUSERS), the number of subsystems (NUMSYSTS), and the number of pages in the time-sharing memory area (USERSIZE) where

NUMUSERS = 3 words per user

NUMSYSTS = 4 words per subsystem

USERSIZE = 2 words per page

Use of Clock 2 (frequency 500 Hz) is required to record the arrival of tasks and their completion.

The calls to BTMSTAT are coded in the BTM Executive with secondary references so that the BTM Executive will not call the Performance Monitor unless it was included at SYSGEN time. The resident portion of BTMPM should be included in the BTM Monitor segment when SYSGEN is executed.

6. PROCESSOR AND SUBSYSTEM FACILITIES

INTRODUCTION

BPM/BTM users may write their own processors and subsystems and use them in addition to or in place of the ones supplied by XDS. User-built processors and subsystems are like any other user program with a few minor exceptions. These exceptions are outlined in this chapter.

A processor is a load module that has been placed in the :SYS account. No restrictions are imposed on the name of a processor except that it cannot exceed 10 characters. Any file in the :SYS account can be called as a processor.

A subsystem is a load module in the :SYS account to whose name is appended a colon character (:). The colon character must be appended by the user at the time the load module is created. At initialization time, subsystems are placed in a reserved area on the swapping device (a user may not put a subsystem directly into that area).

If a new subsystem is added to the :SYS account after loading the system from the PO tape, the subsystem will only become available to on-line users after either a recovery or booting from the system device.

FIXED MONITOR LOCATIONS

For certain purposes, such as the choice of an effective core allocation technique, it is desirable for processors, subsystems, and other programs to be able to identify the Monitor in operation, certain critical locations of the Monitor, and the location of JIT. This is accomplished by having locations 2A, 2B, 4E, and 4F common to all Xerox Monitors. Figure 21 illustrates the contents of these locations.

Location 2A contains a flag that differentiates between an initial boot (nonzero) and a recovery boot (zero).

Location 2B contains three items:

1. Monitor – This field contains the code number of the Monitor. The codes are as follows:

Code	Monitor
0	None or indeterminate
1	BCM
2	RBM
3	RBM-2
4	BPM
5	BTM/BPM
6	UTS
7	Reserved for future use

2. Version – This is the version code of the Monitor and is coded to correspond to the common designation for versions. The alphabetic count of the version designation is the high-order part of the code and the version number is the low-order part. For example, A00 is coded X'10' and D02 is coded X'42'.

3. Parameters – The bits in this field are used to indicate suboptions of the Monitor. They are meaningful only in relation to a particular Monitor. However, the following assignments have been made for BPM, BTM, and UTS.

Bit	Meaning if Set
31	Symbiont routines included.
30	Remote batch routines included.

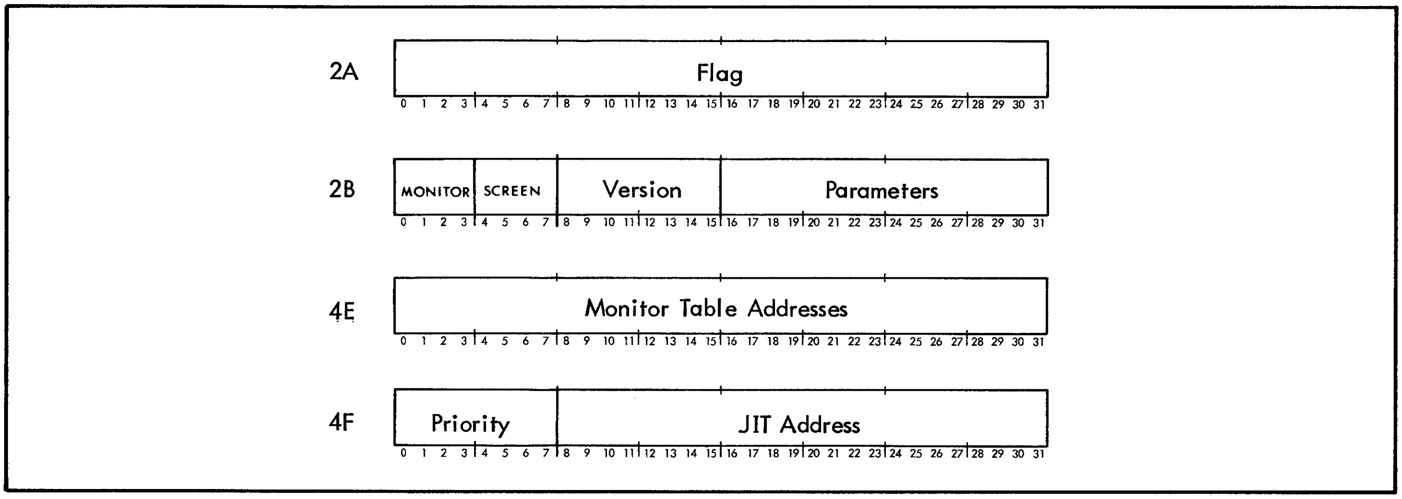


Figure 21. Locations Common to All Monitors

Bit	Meaning if Set
29	Real-time routines included.
28	Unused.
27	Reserved for Data Management System.
26	Unused.
25	Machine has byte string hardware (zero if byte string hardware is not present).

Location 4E contains a pointer to a list of Monitor table addresses.

Byte 0 of location 4F contains the job priority level and the rightmost 17 bits contains the JIT address of the current active job partition. Priority is set to X'FF' for batch and to X'FC' for on-line.

DATA CONTROL BLOCKS

Most processor and subsystem I/O operations are performed through standard Monitor DCBs. For example, source input is normally read by

M:READ M:SI [options]

The standard DCBs are

M:BI	M:SI	M:CO
M:CI	M:C	M:DO
M:EI	M:BO	M:EO
M:LO	M:AL	M:SL
M:SO	M:LL	M:GO
M:PO	M:OC	M:CK
M:LI	M:EF	

The default assignments for batch operations differ from those of on-line operations. This is done so that a program that writes through LO and reads through SI will automatically use the line printer and card reader for batch operations and the terminal for on-line operations. The logical functions associated with the operational labelsare described in the BPM/BP, RT Reference Manual, 90 09 54.

Details concerning input buffers, error handling, and so on are specified as parameters in a read or write call. Parameters associated with files and devices are specified by the ASSIGN control command.

A processor or subsystem may construct its own DCBs by means of the M:DCB procedure. A DCB is limited to 255 words. However, processors, subsystems, or users are not required to have their own DCBs. DCBs not present will be constructed by the Loader. DCBs constructed by the Overlay Loader occupy 48 words. These include space for a 3-word file name, a 2-word account number, a 2-word password, a 3-word INSN, and a 3-word OUTSN. The Lope Loader merges in the ASSIGN information at load time. The Overlay Loader does not perform this function. Assign-merge is performed at execution time by PRGMLDR.

DCBs are also provided in library form and may be explicitly called during a load. Note that library DCBs are not available to the on-line user, or when the Lope Loader is invoked.

Processors or subsystems may use nonstandard DCBs, if necessary. Nonstandard DCBs are constructed by the Loader if not constructed by the processor. They must be explicitly connected to a device either by an M:OPEN call in the processor or by ASSIGN command issued by an on-line user, since no default assignment via operational labels is provided.

It is common practice for a processor, subsystem, or user to obtain source input through M:SI, to print a source listing through M:LO, and to print diagnostic output through M:DO. However, I/O operations are complicated by the fact that an on-line user can connect SI, LO, and DO either to different devices or to the same device (the on-line default assignment for SI, LO, and DO is the terminal). In particular, an on-line user may connect two or more of these standard operational labels to the same device. For this reason, processors and subsystems must take precautions to avoid duplications in printed output. This means that they must know at all times whether they were called in batch or in on-line mode, and what specific device connections have been made for standard DCBs.

Processors and subsystems may examine DCBs directly to determine when the DCBs are connected to the same device. Fields within a DCB may be referenced relative to the name of the DCB. Fields that may be useful to processors and subsystems are as follows:

Field	Use
FCD	Bit 10 of word 0 of a DCB. This is the file-closed flag. A1 means the associated file is open; a 0 means the file is closed.
TYPE	Bits 18-23 of word 1 of a DCB. These bits specify a code for the type of device connected to the DCB (printer, Teletype, card reader, etc.).
DEV	Bits 24-31 of word 1 of a DCB. These bits specifyan indexto the Monitordevice table.

FILE IDENTIFICATION

Most processors and subsystems use a common format and common character set for constructing file identifiers (fid). The standard format is

name [([account][, password])]

where name, account, and password consist of character strings with maximum lengths of 31, 8, and 8, respectively. Any of the following characters may be used:

A-z 0-9 ␣ $ * % : # @ -

CCI SCAN

On transferring control to a user's program, a processor, or subsystem, the Monitor communicates the following information via the general registers:

General Register	Information Communicated
0	TCB address

Processors and subsystems may fetch the card image of the command that called them by reading through a DCB connected to the C device. Alternatively, they may examine, in place, the image as read by the Monitor by using registers 2 and 6 as follows:

General Register	Information Communicated
2	Address of the first word location of the control command and buffer.
6	Byte position (within the control command buffer) of the first byte following the name of the processor or subsystem.

Example:

Assume that the control command buffer (address CCBUF) contains the following data:

CCBUF	!	ƀ	S	Y	word 0
	M	B	O	L	word 1
	b	L	O	,	word 2
	B	O	ƀ	ƀ	word 3
	ƀ	ƀ	ƀ	ƀ	word 4
	⋮			⋮	
	ƀ	ƀ	ƀ	ƀ	word n

In this example, register 2 contains the address of location CCBUF and register 6 contains the number 8.

When running in batch mode, the processor must read the C device once to clear the control command. This command is transferred to the processor's buffer to allow it to examine parameters.

TERMINAL I/O

An on-line user may direct output to his Teletype at any time during execution of a processor or subsystem. Similarly, portions of the input to a processor or subsystem may come from a Teletype. In general, Teletype I/O is the same as other I/O in its use of M:READ and M:WRITE operations and the standard abnormal and error situations. However, Teletype I/O has some features that are significantly different from those for other devices. Some of the differences require special attention by processors and subsystems, but the interface is designed in such a way the processors and subsystems will not have to know whether or not I/O operations are via Teletype, providing they observe certain conventions. On terminal I/O, like all I/O, the user should note that byte displacements remain in effect until replaced, once they have been given. The special problems associated with Teletype I/O are outlined in the following paragraphs.

END CHARACTERS

On input from a Teletype, each record read is terminated by an end character (CR and LF). The end character, if any, is included in the actual record size (ARS) count reported in the DCB (bits 0-14 of word 4). Each processor or subsystem must interpret the different end characters. Processors and subsystems do not have to know that input is via Teletype, provided they treat these characters as terminators and use ARS to determine the actual record received.

Source files for all processors and subsystems, including those in batch operations, may have been prepared on-line. Since records prepared on-line are variable length, it may no longer be assumed that input records are 80-byte card images.

All characters received from terminals, no matter of what type, are translated to the standard EBCDIC character set. The hexadecimal codes for EBCDIC characters are listed in Appendix C.

WRITE OUTPUT

The length of each output line is specified by the SIZE parameter in the M:WRITE procedure call. Carriage return or NEW LINE characters do not terminate a message.

CARRIAGE RETURN

A NEW LINE or carriage return sequence, as appropriate to the type of terminal, is appended to the character string supplied by each write. Thus, under ordinary circumstances, carriage return characters will be supplied when output consists of one line per write and the DCB is connected to a terminal.

ABNORMAL CONDITIONS

If unknown operations are requested of the COC routines (e.g., write end-of-file), they are ignored.

FORMAT CONTROL

It is sometimes necessary to print a line with special spacing or without a carriage return. Processors and subsystems can obtain vertical carriage control by means of two parameters (SPACE and VFC), both of which can be set by the M:DEVICE CAL. The SPACE and VFC parameters have the following interpretations for Teletypes:

Parameter	Meaning
SPACE	If this parameter is set and VFC is not on, the number of spaces indicated minus 1 is inserted before each write. Counts of 0 and 1 result in single spacing.
VFC	If this flag is set, the COC routines simulate the printer's vertical format control as specified in the first character of the text lines written. The simulation is limited to one of the following cases:

Hex. Code	Action
C1-CF	COC inserts 1-15 spaces before printing.
F1	COC skips to top-of-page by skipping four lines and printing the heading information followed by the print line.
60, E0	COC does not insert CRLF after the print line (suppress space).

For BPM (only), information in the page heading may be specified by the user by means of the HEADER and COUNT device CALs. Heading information is taken from the DCB through which the read or write was given. The automatic page heading occupies one line and contains current time, date, user name and account number, user identification and line number, page number, and possibly an administrative message. Headings specified in the DCB of the read or write are produced after the automatic heading with position, text, and page number as specified in the BPM/BP, RT Reference Manual, 90 09 54. The page count in this heading is that carried in the DCB and is reset with each COUNT device CAL. The page count for the automatic heading is carried in JIT and is never reset. The automatic heading is suppressed if the page length is less than eleven lines.

The ESC I sequences (TAB) is removed from Teletype input lines. Blanks are inserted to the next tab location in the input line. The user must have previously set tab stops using the Executive TABS command. If no tab stops have been entered, or the tab stops have been exhausted for the current input record then a ? is echoed to the user, and one blank is inserted in the input record.

Tabs must be specified in ascending order beginning with tab stop position 1. Note that this is different from the line printer tabbing, where the tabs need not be in ascending sequence. Tabs can be set at any time for any DCB. Tabs typed by an on-line user are simulated at the user's console according to the tab settings in the user's context area.

If the ESC RUB sequence is typed at the terminal, the preceding character is removed from the input buffer unless the previous character was an activation character. The user cannot backspace beyond an activation character.

A program can request control when the user presses the BREAK key by means of the M:INT procedure. Whenever the user presses the BREAK key, the program environment at the time of the break is recorded in the user's pushdown stack in his TCB. Execution can be returned to the location following the interrupted instruction by execution of the M:TRTN procedure. A program can return break control to the BTM Executive by executing the M:INT procedure with a break routine address of zero. The break routine address is checked by the Monitor to guarantee that the address lies within the memory allocated to the user. Even if a processor or subsystem has obtained break control, an on-line user can return execution control to the Executive by executing an ESC ESC sequence.

As a safety measure to protect the user against faulty programming in break control routines, the number of times the BREAK key is pressed by a user without intervening characters is recorded. When the count reaches four, control is sent to the Executive as if ESC ESC had been pressed. Thus, the user at the terminal will never find himself locked out. The count of four allows processors (e.g., FDP) to make special interpretations on two or three breaks in a row.

FILE EXTENSION

BPM allows file extension for background jobs. File extension is a convention by which records are added to an output file by successive job steps. Each time the file is opened, the file pointer (tape, disk pack, etc.) is positioned to a point immediately following the last record in the file. Thus, when additional output is produced it is added to the previous contents of the file, thereby extending it. File extension simulates output to physical devices, such as line printers or typewriters, when output is actually directed to a file.

File extension takes effect at the time BPM/BTM opens system output DCBs. The output DCBs that are affected by file extension are those that are currently assigned to files, although normally assigned to devices. They include: M:LO, LL, DO, PO, BO, SL, SO, CO, AL, EO, and GO.

File extension is discontinued when a file is reassigned with an ASSIGN command or when a file is opened with an OPEN procedure call that specifies an explicit file name. In these cases, a new file is created. Extension of the GO file is terminated following the next !JOB control command.

CREATING SUBSYSTEMS

All subsystems and libraries peculiar to BTM are introduced through use of the BPM file manage and load functions, and are independent of the actual SYSGEN process in the sense that they can be added at any time.

The user must create a :BLIB file under the :BTM account.

The LOAD subsystem uses the :BLIB file in the :BTM account as the library, although an option exists for specifying others. Currently, the library file contains only the on-line version of the FORTRAN IV-H run-time, along with the standard mathematical routines.

The on-line version of the FORTRAN IV-H run-time is obtained by replacing the following routines in the standard deck set-up (which assemble either for BTM, BCM, RBM, or BPM) with the BTM versions.

Cat. No.	Module
704216	BF:SV
704293	BF:SC
704294	BF:SW
704295	BF:GCOMS
704305	BF:DIAG
704306	EXIT
704307	BF:SP
704308	BF:SO
704309	BF:MXXX
704310	BF:TSNUM
704334	BF:NLOC
704387	BF:RUNIO
705293	BF:EXIT

A user wishing to create a library that will be used by the LOAD subsystem can do so by using LOPE to create a :BLIB file in his account.

For example:

```
!JOB ACCT, A, 1
!ASSIGN M:EO, (FILE, NEWLIB)
!FMGE (ENTER)
:
ROMs
:
!ASSIGN M:BI, (FILE, NEWLIB)
!LOPE (PERM, LIB)
!FIN
```

The LOAD subsystem can then be directed to use this library by including the following library specification in the option list.

```
OPTIONS:    U(ACCT)
```

A subsystem is defined by loading it under the BPM system account (:SYS) and forming a load module with the name ending in a colon (:) and a bias equal to the base of the on-line memory area. For example:

```
(LMN, SYMBOL:)
```

The load module must conform to the interface rules laid down in the next two sections.

When the BTM Monitor is booted from disk, it searches the system account for all load modules with this naming convention and incorporates them as on-line subsystems.

The first two characters of the name become the Executive command by which the subsystem may be called, and the remainder of the name (exclusive of the colon) is echoed by the Executive. For example:

```
!SY MBOL
```

When the system is booted, the subsystem load modules are copied to a dedicated area of the RAD or disk pack and referenced from there in absolute format.

The Executive service routines called by the ASSIGN, RESTORE, SAVE, and TABS commands are part of the resident Monitor and therefore do not have corresponding load modules.

SUBSYSTEM CODING REQUIREMENTS

The first 40_{16} locations in any subsystem should contain the following data.

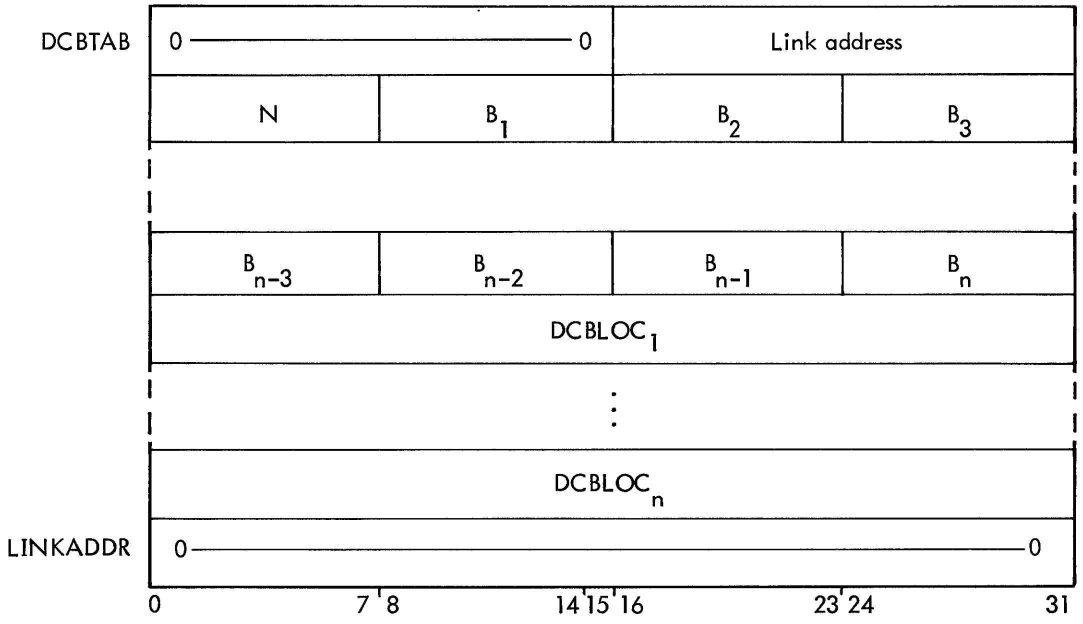
Location	Data Description
0	Contains the word address of the first word of the subsystem's TCB (Task Control Block).
1	=0
.	.
.	.
.	.
8	=0

Location	Data Description
9	Contains the word address of the normal entry point for the subsystem. The entry point for a "PROCEED" is assumed to be this address plus one.
A_{16}	=0
.	.
.	.
.	.
F_{16}	=0
10_{16}	Reserved for use by Monitor.
.	.
.	.
.	.
$3F_{16}$	Reserved for use by Monitor.

When a subsystem is entered, R1 contains the COC line number (in binary), R4 and R5 contain the log-in account designation (in EBCDIC, left-justified and blank filled), and R2 contains the terminal job entry flag (0 indicates that the console is excluded from the system, and a value of 1 - F indicates the maximum priority). R3 will contain the batch authorization flags from AJIT in byte 0. R13 through R15 will contain the LOGIN name designation (in EBCDIC, left-justified, and blank filled).

Included with the ROMs that constitute a subsystem there must appear a DCB name table (see Figure 22) pointed to by word 10 of the TCB (see Figure 23), and all the necessary DCBs assembled with protection type 00.

There are several CALs available (for use by a subsystem only) to provide service for the subsystems. These CALs mainly deal with changing the size of swap areas for subsystems and users, finding out the amount of core currently being used by subsystem and user, and performing swaps between user and subsystem memory. They are described in the BTM/TS Reference Manual, 90 15 77. The format for DCBs is essentially the same as described in the BPM/BP, RT Reference Manual, 90 09 54. The one difference is the ability to assign DCBs to a user's console. This is done by setting the ASN (bits 28-31) in word 0 of a DCB to 5 (ASN is the file assignment type indicator — 0 means null, 1 means FILE, 2 means LABEL, 3 means DEVICE, 5 means user's console).



where

LINKADDR is the address of the location provided for storing a return address.

N indicates the number of characters in the DCB name.

B_1 - B_n indicates the EBCDIC name of the DCB.

DCBLOC is the address of the first word location of the DCB.

Figure 22. DCB Name Table

0 7 8 14 15 16 23 24 31

Word 0	0——————0	TSTACK	
1	TSS	TSWC	
2 5	These words for use by subsystem		
6	0——————0	TSA	
7	TSASIZ	TSAWC	
8	ERTSIZ	ERT	
9	ERTSIZ-2	TSA + 1	
10	0——————0	DCBTAB	
11 15	These words for use by Monitor		
TSA	Library error temp stack		TSASIZ
ERT	Library error table		ERTSIZ
TSTACK	User's temp stack		TSS

0 7 8 15 16 23 24 31

where

TSTACK is the address of the current top of the user's temp stack. Initially, TSTACK points to the start of the stack minus one.

TSS indicates the size, in words, of the user's temp stack.

TSWC is the temp stack word count giving the current number of words in the user's temp stack.

TSA is the address of the temp stack used by the library error package.

TSASIZ indicates the size, in words, of the temp stack used by the library error package.

ERTSIZ indicates the size, in words, of the error table used by the library error package.

ERT is the address of the error table used by the library error package.

DCBTAB is the address of a table of names and addresses of all of the user's DCBs.

Figure 23. TCB Format

SUBSYSTEM LOADING REQUIREMENTS

The ROMs that comprise the subsystem should be loaded into the system by running the following job under BPM.

```
!JOB        :SYS, user, 1
!LOAD       (ABS), (BIAS, loc), (NOTCB), (PERM), ;
!           (LMN, name:)
            .
            .
            .
            subsystem ROMs
            .
            .
            .
!FIN
```

where

name is the subsystem name to which console users will refer (followed by the colon character).

loc is the hexadecimal value of the base of the on-line memory area:

When the system is booted from the RAD, the load module will be initialized in absolute swap storage as a subsystem callable by the first two characters of the name.

7. MONITOR DUMP PROCESSOR

INTRODUCTION

The Monitor Dump processor (MONDUMP) is designed to aid the system manager in the debugging of BPM/BTM crash dumps. It performs this function by listing the contents of a crash recovery file or tape in as meaningful a way as possible. MONDUMP is called in for execution by the control command

```
!MONDUMP
```

followed by one to eight control commands that specify the region(s) to be dumped and the output format of the listing. If no commands follow the MONDUMP command, default options (described below) are given.

JOB SETUP

Unless the user wishes to take the default options, MONDUMP requires two kinds of information in order to execute. It must know what tape or file to list and what the format of the output should be.

INPUT SOURCE

An ASSIGN card is used to specify the input before the MONDUMP command is encountered. The assignment is to the M:BI (not M:EI) DCB. Start addresses of TAPE and FILE are not included. All input source information is obtained from the ASSIGN card. If no ASSIGN is given, the default is the MONDUMP file (:BTM). The ASSIGN card must include an account number even if the job account is intended; otherwise, :BTM would be assumed.

The following example lists a crash recovery tape:

```
!JOB :SYS,LIST-CRASH,F

!ASSIGN M:BI,(DEVICE,9T),(SN,ANY)

!MONDUMP
```

LISTING FORMATS

MONDUMP permits the user to specify up to eight regions to be dumped. Each region must have a format specification. The cards containing the specification follow the MONDUMP processor call; no options are honored on the processor card. If there are no valid control cards following MONDUMP or an end-of-file is encountered immediately (under BTM, a blank card is also an end-of-file), the default listing options are taken. This default is described below.

Individual control cards have the format:

```
X,Y,Z
```

where

X,Y are the hexadecimal numbers indicating the start and end regions. It is the user's responsibility to ensure that these addresses are valid core addresses. They will automatically be adjusted to 8-word boundaries. Leading zeros are optional but leading blanks are not; a blank is treated as a comma.

Z is one of the numbers 4, 6, 8, or 12. This field specifies the format in which the region is to be printed. The numbers indicate how many words are included on a print line. As fewer words are included on the print line, more information is displayed for each location.

The following matrix shows what information is included:

		Hex.	EBCDIC	Opcode	DEF Name
Format	4	Yes	Yes	Yes	Yes
	6	Yes	Yes	Yes	No
	8	Yes	Yes	No	No
	12	Yes	No	No	No

The convention followed for the EBCDIC translation is that characters in the Xerox standard 63-character graphic set that will not print on the standard line printer (such as !) are replaced by %. The convention followed for op codes is that they will be suppressed if the four bytes comprising the instruction word are all printable or if it is unlikely that a word is an instruction. Suitable mnemonics are substituted for op codes whenever possible. The DEF name applies only to the Monitor root. It is the external name truncated to eight characters.

There are no restrictions placed on addresses or formats by the control card processor. Thus, the control commands could be used to bypass the Monitor (possibly applicable in a real-time system), to give an all-hex dump of areas of little concern (e.g., background or the recovery overlay), or to bypass the dump portion altogether so that only a summary results.

ERROR MESSAGE

The extension of significant information beyond card column 15 is an error. The following error message will be listed:

```
***ABOVE CC IN ERR, IGNORED-MDSUPER***
```

Note that all MONDUMP error messages have the name of the issuing module included (MDSUPER, in this case). The above error message can occur for several other reasons, such as illegal characters (G-Z), more than eight control cards, unrecognized format type, etc.

DEFAULT FORMATS

If no valid control cards are found, the default listing formats apply; otherwise, no defaults apply. The default listing format is 4 for the root of the Monitor and 8 for the rest of core. If MONDUMP is unable to access the M:MON (:SYS) file, the default is 8 for all of core (it is a user responsibility to avoid using the 4 specification in the latter event).

CRASH ANALYSIS

MONDUMP performs a significant amount of processing on the dump to display meaningfully what has happened. Note that none of the features listed below are available unless MONDUMP is able to access the M:MON file under :SYS.

- The software check number and description are listed. The contents of the general registers are reconstructed as completely as possible at the time of the trap or error, rather than upon entry to recovery. Explanatory messages with each software check tell which registers are valid and which are lost.

- The active user is determined. The following items are listed for the active user:

Active User	Name	Acct.	Job ID	Line No.
BPM	Yes	Yes	Yes	N.A.
BTM	Yes	Yes	N.A.	Yes
RTIME	N.A.	N.A.	N.A.	N.A.

This information about the active user is also listed on the operator's console.

- The interrupt and trap locations are analyzed. For each trap, the new and old PSD are listed. The contents of the trapped location is listed. If the trapped location is illegal (within registers or above core), a message is printed. Similarly, if the trapped location was overlaid by recovery, the fact is noted. The names for the PSD and the PSD handler are listed.

- All of the DCBs in the system are analyzed. The name, assignment type and location of each DCB are printed. If TYC was normal, the OPEN/CLOSE status of the DCB is printed; otherwise, a message explaining the TYC is listed. Several integrity checks are made on the DCB tables.

- The overlay structure is determined from COVLSEG and S1TB. For symbiont systems, a message will be printed if the CPOOL, SPOOL, or MPOOL chains are zero.

- The DCT and IOQ tables will be dumped. Each entry will be identified both by DCT/IOQ number and by a short mnemonic. Thus, the heading for DCT2 would be

```
 2
ADR
XXX
XXX
XXX
```

Most entries are listed in hexadecimal, exactly as found in the tables, with the following exceptions:

DCT4 The type mnemonic rather than the type mnemonic index is listed.

DCT7 This is listed as a word address rather than a doubleword.

DCT16 This is translated to EBCDIC following standard MONDUMP conventions (the N/L is printed as a "." and the exclamation as "%").

IOQ6 The software function code is listed with IOQ4 and 5. The DCB address is listed separately.

IOQ7 The EBCDIC translation of the DCT index is appended (e.g., 03-.%%CRA03 rather than 03).

- The symbiont tables parallel to SNDDX are listed for symbiont systems.

- The BTM parallel tables indexed by the user are listed; the subsystem tables are omitted. The PSD information is not printed in its entirety. Only PSW1 and the error code in PSW2 are given.

- A dump of INBUF, OUTBUF, and COCBUF1 in the BTM COC buffer is given. Both the byte and word addresses are carried. The translation is in 7-bit ANSCII, with valid ANSCII characters not printing on the standard 56-character printer set represented by %. No duplicate line suppression is done.

DUMP TAPES

It may not be known at the time a system recovery is effected wherether an analysis of the dump should be performed. A method for saving several binary dump images on a single tape is given below.

During SYSGEN, the M:MON (:SYS) file should be written out to a labeled tape as the first file. Having the file on tape permits dumps on it to be run under different versions of the Monitor or on a different machine. Further, the M:MON (:SYS) file need not be made permanent, since its temporary restoration can always be made part of the job stack that reads files from the tape.

Whenever a recovery occurs, the labeled dump tape with M:MON on it is positioned to the end of all files using the FMGE control command as follows:

```
!FMGE
```

```
!ASSIGN M:EI,(LABEL,xxx),(SN,MOND)
```

The dump is written to the tape after the current end of all files. Note that "INSN" and "OUTSN" can be replaced by "SN".

With such a tape, it is possible to list the dump at any time, at any installation with the deck setup given in Figure 24.

At the time of a crash, a quick summary can be obtained and the dump image can be saved on tape. Later, when it is determined if anything of interest is in the dump, appropriate control cards can be made up, thus permitting the installation to be selective as to what dumps are listed and how much time and paper are expended.

MONDUMP ASSEMBLY OPTIONS

Normally the following assembly options should not be necessary. If the user requires options other than the defaults provided, these are described below.

SIZE OF MONDUMP

The module MDSYMTAB contains a RES large enough to accommodate the record MON: :ORG in the file M:MON. If the RES for MDSYMTAB is too small, the following control commands could be used in determining how large this record is:

```
!FMGE (LIST,BIN)
```

```
!ASSIGN M:EI(FILE,M:MON,:SYS)
```

The byte count of the MON: :ORG record in the resulting listing is then examined.

DEFRES is large enough to accommodate most large systems. As a very crude measure, DEFRES should be about 40 percent of the size of the Monitor to be listed. By the appropriate setting of DEFRES and re-assembling MDSYMTAB, unusual situations may be accommodated.

It should always be possible to run MONDUMP. For instance, assume a 32K BTM system with a 12K user size/swap area (the limiting, supported configuration). If the remaining 20K consisted of a Monitor, then DEFRESZ should be about 8K. MONDUMP will fit in 12K and reserves more than enough space for the approximate 8K record.

If an installation is running a restricted, non-supported version of BPM or cannot make the M:MON file available, DEFRESZ can be set to 2K. A smaller figure could not be used since MONDUMP packs the DEFs it reads and reclaims the space freed by this procedure for other buffers. Setting DEFRESZ to 0 would cause a misallocation of these other buffers.

SIGMA 7 INSTRUCTIONS

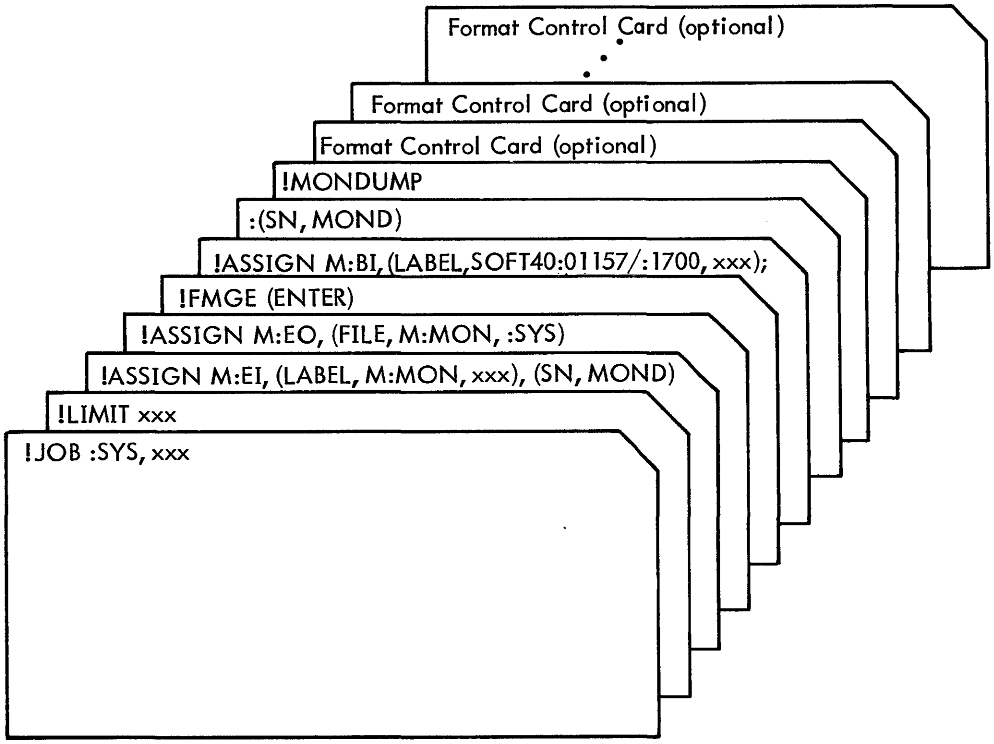
As distributed, MONDUMP is assembled for Sigma 5. If MONDUMP is run on a Sigma 7, it may be reassembled to utilize byte string instructions as follows:

```
+116,116        (in MDSYSTEM)

$SIGMA   SET         7

+END
```

This assembly will speed up operation and slightly reduce the size of the program.

ASSEMBLY LISTING FORMAT

In most cases, the code listed by the PROCs will not be listed. Rather, the current starting address of the $PSECT1 control section is displayed on a single line regardless of



```
!JOB :SYS,xxx
!LIMIT xxx
!ASSIGN M:EI,(LABEL,M:MON,xxx),(SN,MOND)
!ASSIGN M:EO,(FILE,M:MON,:SYS)
!FMGE (ENTER)
!ASSIGN M:BI,(LABEL,SOFT40:01157/:1700,xxx);
:(SN,MOND)
!MONDUMP
Format Control Card (optional)
Format Control Card (optional)
   •
   •
Format Control Card (optional)
```

Figure 24. Deck Setup to List Dump

the amount of code generated by the PROC. A listing of the code could be obtained as follows:

```
+85,85          (in MDSYSTEM for all modules of
                 MONDUMP)
$LISTLVL   Set   50

+END
```

or

```
+(past the MDSYSTEM call in any module only for the
                     module currently assembling).
$LISTLVL   SET   50

+END
```

PROCs that normally generate only a single word of code will always display the word generated.

The individual modules of MONDUMP must be assembled with the MDSYSTEM file in the :SYS account. MDSYSTEM is a Meta-Symbol system file such as MON, BPM, SIGMET, and SIG7FDP. If listings of MONDUMP are desired, it is strongly recommended that MDSYSTEM also be assembled as follows:

```
!ASSIGN M:CI (FILE,MDSYSTEM,:SYS)

!METASYM CI,LO,CN

.END

+END
```

All of the PROCs used in writing MONDUMP are included and documented in MDSYSTEM. A complete assembly-load of MONDUMP takes approximately 30 minutes.

PATCH AREA

Although not strictly an assembly option, 20 unused locations, starting with the DEF "PATCHES", are provided for making patches and GENMODS to MONDUMP.

8. BPM ERROR LOGGING (ERRLOG AND ELIST) ROUTINES

INTRODUCTION

The BPM Error Logging (ERRLOG) Routine packs error messages into a buffer and writes the buffer to a special consecutive file on disk. The ERRLOG Routine is called from the various parts of the BPM system that detect errors such as I/O errors, memory parity, etc. When an error is detected by the BPM system, it constructs a message and then calls the ERRLOG Routine which packs the message in a buffer. Messages may be up to ten words in length and may be in registers or in memory.

The Error Log Lister (ELIST) reads information from the packed buffers, and from files created by ERRLOG, and reformats the messages into a meaningful listing. ELIST runs as a background program using the I/O CALs provided by the Monitor.

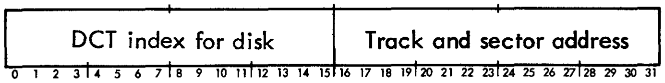
ERRLOG FILE FORMAT

The ERRLOG File is a special consecutive file on the disk. The file is not accessed by name, but by a special pointer in memory which contains the disk address of the first record in the file. Each record is 256 words long and contains a backward and forward link. The backward link is the disk address of the previous record and the forward link is the disk address of the next record in the file. The backward link in the first record of the file is zero. The forward link is never zero since the file can never be considered as complete. As long as the system is running, the file is considered as open and the next record to be written is being constructed in a core buffer as the errors occur. The forward link in the last record written contains the disk address where the next record is to be written.

The record format in the file is:

0	Backward Link (BLINK)
1	Forward Link (FLINK)
2	Number of words of message
3	Message 1
	Message 2
255	Message 12

Each record may contain several unused words at the end of the record since messages may vary in size, from one to ten words each. Therefore, up to nine words may be unused. The maximum number of useful words in a record is 253. The format of the backward and forward link disk addresses is shown below. This format is used for all disk addresses.



Disk granules for writing the file are obtained by calling the Get Background Granule (GBG) routine. This routine gets one granule at a time. One granule consists of two consecutive sectors on a disk that has 256-word sectors and six consecutive sectors on a disk that has 90-word sectors. Therefore, each granule contains two records of the file.

ERRLOG FILE CONTROL POINTERS

Four pointers in memory provide the necessary control information for constructing and accessing the ERRLOG File. The following is a list of 1-word pointers and their definitions.

SGRAN	Contains the disk address of the first record in the file.
BGRAN	Contains the disk address of the last record written.
CURGRAN	Contains the disk address of the next record to be written.
FGRAN1	Contains the disk address of the record to be written following the current record.

One 256-word buffer is used for packing the error messages. The buffer is used as a push-down stack having 253 words. The error message is pushed into the stack and if fewer than 10 words are remaining, the buffer is written to secondary storage. The forward and backward link disk addresses and the number of useful words are put into the buffer just prior to writing it to the disk.

If the buffer is in the process of being written to secondary storage and a message is sent to the ERRLOG routine, the routine will wait for up to five seconds for the I/O to complete, and then exit back to the user thus losing the message. If the I/O completes during the five seconds, the message will be put into the buffer normally.

Programs that operate outside the Monitor may access the file control pointers in the following manner. Location X'4E' contains a pointer to a vector table containing a pointer to the file control pointer table, as shown below.

```
X'4E'                         DATA    CCITAB
CCITAB + 0                    DATA    - -
       + 1                    DATA    - -
       + 2                    DATA    - -
       .
       .
       + 15                   DATA    ERRLGTAB
ERRLGTAB + 0                  DATA    BUF1
         + 1   SGRAN          DATA    disk adr
         + 2   BGRAN          DATA    disk adr
         + 3   CURGRAN        DATA    disk adr
         + 4   FGRAN1         DATA    disk adr
```

In addition, a program operating from the :SYS account may have access to the ERRLOG File through CAL1,6, which reads the first buffer full of information from the file and returns the disk granule to the Monitor's available pool when both buffers from the granule have been read. The FPT format is

Word 0

X'00'	Address of place to put information
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

If the user's account is not :SYS, no action is taken and CC1 is set to 1.

Normal returns set all condition codes to zero. When the final record has been read from the ERRLOG File (including any partial records from memory), CC3 is set to 1.

If the ERRLOG File cannot be read without error, then CC2 is set to 1, and whatever has been read is transferred.

If the error log buffer is empty and no records are on disk, CC4 is set to 1.

For purposes of reliability through simplicity, the ERRLOG File does not operate through the standard file mechanism. Thus, the ordinary protections against simultaneous use are not available and a single program should be used to access this file. Also, the ordinary file backup procedure does not apply, so information must be to an ordinary file for final recording.

ERRLOG CALLING SEQUENCE

The ERRLOG routine is called with the address of the message in register 6. The message may be in registers or in memory. If the message is in registers it may wrap around, e.g., the message may start in register 12 and end in register 3. The calling sequence is as follows.

```
BAL,5   ERRLOG
RETURN
```

All registers are nonvolatile.

ERRLOG INPUT/OUTPUT SYSTEM FORMATS

Whenever I/O errors or certain unusual conditions occur, an entry will be made into the ERRLOG file. This entry will contain any information pertinent to the condition.

Word 0 of each entry will have a code indicating which error or unusual condition is present along with the number of words in the entry (including word 0). Time (hhmm) and Device Name (yyndd) are in EBCDIC.

START INPUT/OUTPUT FAILURE

In this case, CC1 or CC2, or both, are set when the Start Input/output, SIO, is issued. The entry has the form shown below.

Word 0

X'01'	X'04'	▨	T
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 1, time

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2

S	C	y
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 3

y	n	d	d
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

where

S is the SIO status (16 bits).

C is the SIO condition codes.

T is the bits 0-7 of the Test Device, TDV, status.

DEVICE TIMED OUT

In this case, a Halt Input/output, HIO, was issued due to excessive time before occurrence of I/O interrupt. The entry has the form shown below.

word 0

0–7	8–15	16–23	24–31
X'02'	X'07'	▨	T

word 1, time

0–7	8–15	16–23	24–31
h	h	m	m

word 2

0–15	16–23	24–31
H	C	y

word 3

0–7	8–15	16–23	24–31
y	n	d	d

word 4 - command doubleword

0–31
word 1

word 5

0–31
word 2

word 6 - applicable to disc only

0–31
Seek address

where

H is the HIO status (16 bits).

C is the HIO condition codes.

T is the bits 0-7 of TDV status.

UNEXPECTED INTERRUPT

In this case, an Acknowledge Input/Output (AIO) interrupt has occurred for a device which was not busy. The entry has the form shown below.

Word 0

0–7	8–15	16–31
X'03'	X'03'	▨

Word 1

0–7	8–15	16–23	24–31
h	h	m	m

Word 2

0–31
AIO data

NO INTERRUPT RECOGNITION

In this case, the AIO condition codes indicate no recognition. The entry has the form shown below.

word 0

0–7	8–15	16–31
X'04'	X'02'	▨

word 1, time

0–7	8–15	16–23	24–31
h	h	m	m

DEVICE ERROR

In this case, an error or certain unusual conditions have been detected when attempting to use the device. The entry has the form shown below.

word 0

0–7	8–15	16–31
X'05'	X'07'	T

word 1, time

0–7	8–15	16–23	24–31
h	h	m	m

word 2

0–7	8–23	24–31
A	▨	y

word 3

0–7	8–15	16–23	24–31
y	n	d	d

word 4 - command doubleword

0–31
word 1

word 5

0–31
word 2

word 6 - applicable to disk only

0–31
Seek address

where

A is the AIO status (16 bits).

T is the TDV status (16 bits).

MEMORY PARITY

When a memory parity interrupt occurs, all of the memory will read with a Load Word instruction. The first two bad locations plus a count of the total number of bad locations will be logged, along with the addresses of any I/O devices active at the time of the interrupt (up to a maximum of six, due to space considerations; empty slots will be filled with zeros).

After logging the error, the message

```
!!MEMORY PARITY ERROR
```

will be typed and the Wait state will be entered with all interrupts inhibited. The entry has the form shown below.

word 0

X'07'	X'01'	Fault indicator
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2

PSD word 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 3

PSD word 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 4

Bad location 1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 5

Bad location 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 6

Total number of bad locations
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

words 7-9

Active device addresses
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SYSTEM START-UP

word 0

X'08'	X'04'	Recovery Count, 0 to n
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1, time

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2, date

m	m	d	d
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 3, date

ƀ	ƀ	Y	Y
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

WATCHDOG TIMER RUNOUT

word 0

X'09'	X'04'	0	0
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 1, time

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2

PSW1
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 3

Instruction
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where

PSW1 is the contents of PSW1 at Watchdog Timer Runout.

INST is the instruction indicated by PSW1.

FILE CONSISTENCY CHECK FAILURE

word 0

X'0A'	X'03'	Error and Sub Code
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1, time

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2

DCT Index	Relative Sector Number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where Error and Subcode, and Relative Sector Number are the disc address containing erroneous information.

SYMBIONT CONSISTENCY CHECK FAILURE

word 0

X'0B'	X'03'	DCT Index of Symbiont
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

word 1, time

h	h	m	m
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23	24 25 26 27 28 29 30 31

word 2

DCT Index	Relative Sector Number
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

where DCT Index and Relative Sector Number are the disk address containing erroneous information.

ERROR LOG LISTER (ELIST)

The Error Log Lister (ELIST) can be assembled into one of two versions.

1. Stand-Alone Routine. This version should be used if the BPM/BTM system recovery is impossible.

2. BPM Processor. This version should be used if the BPM/BTM system is operational.

The differences to be considered between the two versions are shown in Table 3.

BPM PROCESSOR FOR ELIST

The BPM Processor version is preferred for periodic examination of the ERRLOG file. There are no I/O recovery restrictions in this version because it operates under control of the Monitor.

BPM PROCESSOR CALLING SEQUENCES

The BPM processor is called by a control command of the form

```
!ELIST [(FILE,filename)]
```

where filename specifies the name of an optional file to which a copy of the system ERRLOG file will be added.

If the FILE parameter is omitted, only a listing is produced.

Table 3. ELIST Considerations

Consideration	Stand-Alone	BPM Processor
Scope of listing	Core buffers and ERRLOG file	ELIST copy file, ERRLOG file and core buffers
Order of listing	Most recent first	Most recent last
ERRLOG file copy	Not produced	Keyed file in response to file parameter
Disposition of ERRLOG file	Kept	Destroyed
Loading	Bootstrap	!ELIST control command
Program Bias	X'7000'	Background area
Special Privileges	Master Mode	Job runs under :SYS
I/O Recovery	Restricted	Uses standard Monitor error recovery
Output Device	LPA02	M:DO for listing, F:1 for copy file

A typical job deck for ELIST would be

```
!JOB       :SYS,name,F
!LIMIT     options
!ELIST     (FILE,filename)
```

Jobs must run under the :SYS account because the errors detected by the BPM system are read destructively by ELIST. ELIST runs as a background processor using the Monitor's I/O CALs. If the job does not run under :SYS, a message

NOT RUNNING UNDER :SYS ACCOUNT

is printed and ELIST aborts processing

ELIST COPY FILE

A copy of the errors detected by the BPM system is added to an optional file because ELIST reads destructively. This optional file can be legal system file name specified in the ELIST control command. ELIST uses the F:1 DCB for the copying function. If the file already exists, the records comprising the current error file are added to the existing file.

If the file does not already exist, the file named by ELIST will be created as a keyed file with individual record keys in the format shown in Table 4.

The year, month and day (bytes 0, 1, and 2) are obtained from the System Startup entry. If this entry is not the first in the file, the current data is obtained through the M:TIME procedure.

Table 4. ELIST Copy File

Byte	Keyword	Description
0	Year	Binary representation.
1	Month	Binary representation.
2	Day	Binary representation.
3	Hour	Binary representation.
4	Minute	Binary representation.
5	Error Count	Most significant digit.
6	Error Count	Least significant digit.
7	Error Type	Identical to byte 1 of the error message.

The error count is the hexadecimal representation of the number printed with every message, e.g., ERROR 7. If the number printed is a range of numbers, because several messages are identical, the error count is the first or lowest number of the range.

BPM PROCESSOR LISTING

The BPM Processor uses the M:DO DCB to produce the listing.

The listing is in the chronological order of first entries through to most recent entries. When the listing of the Copy File (optional file), the ERRLOG File, and core buffers is completed, the error summary is produced.

STAND-ALONE ROUTINE FOR ELIST

The Stand-Alone version has two restricted I/O error recovery capabilities.

1. Should a card reader fault occur when booting the Stand-alone deck, the BPM/BTM restart will be activated.

2. An erroneous disk address causes the program to WAIT at symbolic location WRDSKAD.

STAND-ALONE LOADING

Use the standard bootstrap procedures with the restriction:

Do not clear core when loading ELIST.

The most recent ERRLOG information is retained in core and cannot be listed if destroyed.

ELIST will start at location X'7000'.

STAND-ALONE LISTING

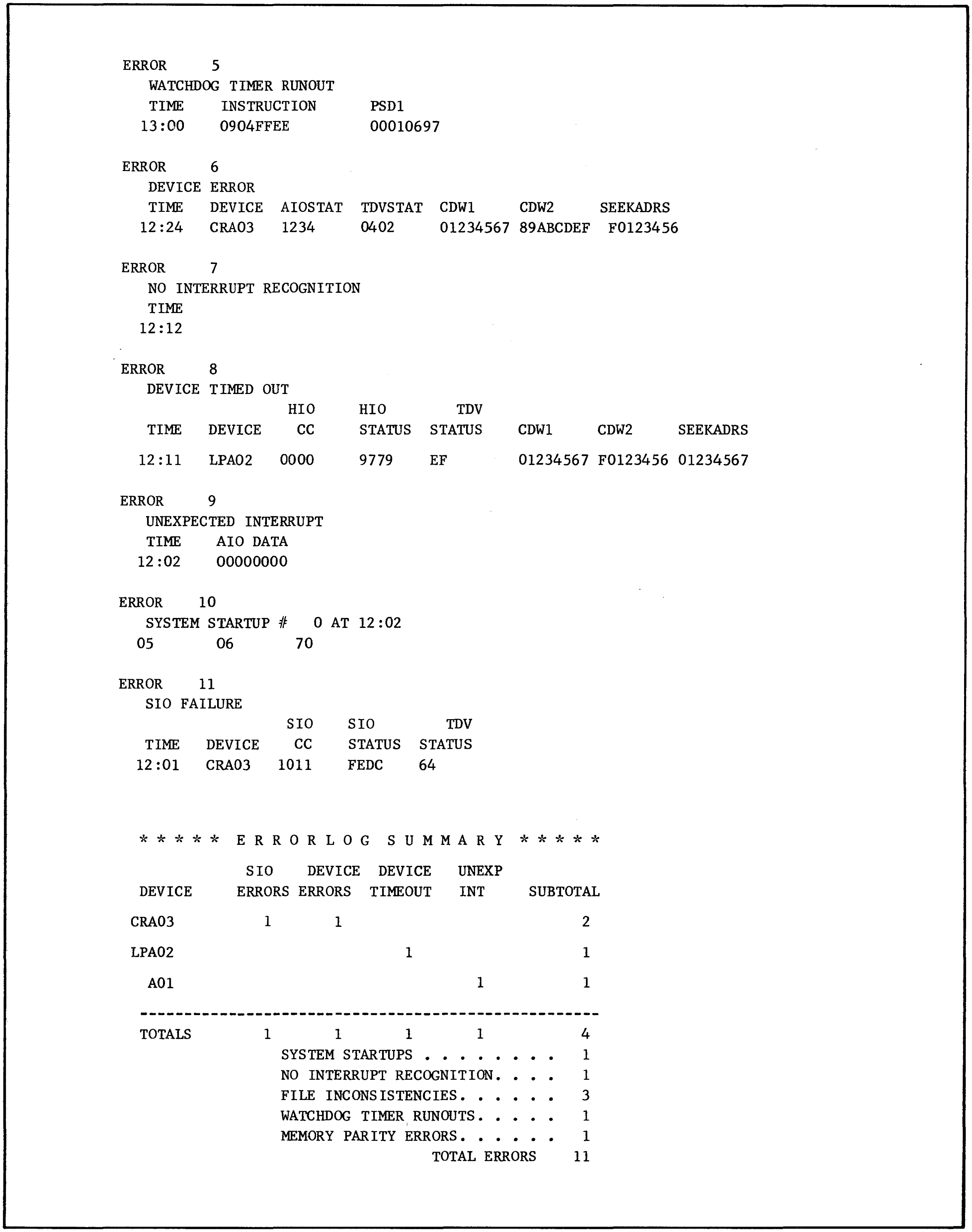
Output will occur on device LPA02. If this is to be modified, symbolic location TYPE will have to be assembled to some other value.

The listing will be in reverse chronological order: the most recent entries will be printed first. For example, the Symbiont File Inconsistency entry will be listed first instead of last, as shown in the "ERRLOG I/O System Formats". Figure 25 shows the format of the ELIST listing. (Note that Figure 25 shows format only; the content of the list is not typical.)

The listing is of the core buffers and the ERRLOG File followed by a summary of the errors.

```
ERROR     1-0002
   SYMBIONT FILE INCONSISTENCY
   TIME    DEVICE  SYMB. DEVICE SEEKADRS
 13:15   LPA02    LPAOF            1117

ERROR     3
   FILE INCONSISTENCY ERROR
   TIME    DEVICE  ABN/SUBCODE  SEEKADRS
 13:13    LPAOF         OF/1B      3333

ERROR     4
   MEMORY PARITY ERROR
   TIME    FAULT   PSD1     PSD2     LOC1      LOC2      TOTAL ACTIVE DEVICES
 12:34    0005    FF000011 EE000022 DD000033 CC000044     3 0001  0002
                                                    FFD4  0000  0005  0006
```

Figure 25. ELIST Listing



```
ERROR     5
   WATCHDOG TIMER RUNOUT
   TIME    INSTRUCTION       PSD1
  13:00    0904FFEE          00010697

ERROR     6
   DEVICE ERROR
   TIME   DEVICE  AIOSTAT  TDVSTAT  CDW1     CDW2     SEEKADRS
  12:24   CRA03   1234     0402     01234567 89ABCDEF  F0123456

ERROR     7
   NO INTERRUPT RECOGNITION
   TIME
  12:12

ERROR     8
   DEVICE TIMED OUT
                   HIO     HIO       TDV
   TIME   DEVICE    CC     STATUS  STATUS     CDW1     CDW2     SEEKADRS

  12:11   LPA02   0000     9779    EF         01234567 F0123456 01234567

ERROR     9
   UNEXPECTED INTERRUPT
   TIME    AIO DATA
  12:02    00000000

ERROR    10
   SYSTEM STARTUP #   0 AT 12:02
  05       06       70

ERROR    11
   SIO FAILURE
                   SIO    SIO       TDV
   TIME   DEVICE    CC    STATUS  STATUS
  12:01   CRA03   1011    FEDC    64


  * * * * *  E R R O R L O G   S U M M A R Y  * * * * *

                SIO    DEVICE  DEVICE    UNEXP
  DEVICE      ERRORS  ERRORS  TIMEOUT    INT      SUBTOTAL

 CRA03           1       1                             2

 LPA02                           1                     1

   A01                                     1           1

  ----------------------------------------------------
  TOTALS         1       1       1         1           4
                        SYSTEM STARTUPS . . . . . . . .  1
                        NO INTERRUPT RECOGNITION. . . .  1
                        FILE INCONSISTENCIES. . . . . .  3
                        WATCHDOG TIMER RUNOUTS. . . . .  1
                        MEMORY PARITY ERRORS. . . . . .  1
                                        TOTAL ERRORS    11
```

Figure 25. ELIST Listing (cont.)

9. FILE ANALYZER

INTRODUCTION

The File Analyzer (FANALYZE) is designed to provide fast and efficient reliability checks on a large BPM/BTM file management system. It verifies the granule pool bit maps, checks all linkages throughout the account and file directories and master index blocks (all levels), and produces a log on the M:LO device with pertinent information concerning the file structures.

FANALYZE is essentially a dual-purpose utility. It allows a general picture of the file system if that is all that is desired. Through the use of options, a more detailed examination can be made if errors are found at any level.

To load FANALYZE, the standard file $::SPECIALS should be selected from the BI tape during SYSGEN PASS1 and the LOCCT name FANALYZE should be included during PASS3 (see Chapters 11 and 12).

FANALYZE OPTIONS

The File Analyze processor is called by specifying the processor name and, if desired, one or more of the options.

```
!FANALYZE [option,...]
```

where option may be any one or more of the following:

VERIFY performs a validity check. If VERIFY is specified without the NOMAP option, a copy of the existing granule pool map is created in the background area. Then all account directories, file directories, file information tables, master index blocks (level 0 through level N) and file data granules are allocated space on a master copy granule map in the background area. The Monitor map is compared against the master map and the two maps are snapped for evaluation if an error is detected (see error definitions). All linkages are checked and linkage failures are reported whenever found.

The VERIFY option will only compare against existing maps. At program completion, it is up to the user to determine from the snaps taken if RECOVERY2 should be run.

NOMAP performs all the validity checks except for the granule pool maps (this is essentially a verify with the map code disabled).

ANLZ (account, [filename]) specifies how much output the analyzer will produce. No granule map work is attempted but the file information table (FIT) is listed and all master index entries are listed. All linkage checks are performed and disk addresses are verified (both index and data locations). Omission of filename will cause a listing for all files in the account.

DATA specifies that all data granules are to be format-listed on the M:LO device at the completion of an analyze pass on one file. This option is valid only when used with ANLZ.

NOLIST specifies that no format listing of file information is to be done. If errors are encountered, the account number and filename are logged before the snap is taken on the M:LO device.

AD specifies that the account directory sectors are to be printed as they are encountered. When used by itself, this option will list all account directory sectors.

FD specifies that the file directory sectors are to be printed as they are encountered. When used by itself, this option will list all file directory sectors.

Examples:

1. This example will verify the existing granule map and list all the names found.

```
!JOB :SYS, USERNAME, F
!FANALYZE VERIFY
!FIN
```

2. This example will list all master index locations and index keys for the COBOL file in the :SYS account.

```
!JOB :SYS, USERNAME, F
!FANALYZE ANLZ(:SYS, COBOL)
!FIN
```

To list all master index locations and index keys for all the files in the :SYS account, the commands would be

```
!JOB :SYS, USERNAME, F
!FANALYZE ANLZ(:SYS)
!FIN
```

INPUT/OUTPUT

Input and output for FANALYZE is performed in the following way:

1. All disk I/O is performed through the Monitor routine NEWQ.

2. All line printer output is performed via CAL1s through the M:LO DCB.

3. The processor control command is read through the M:C DCB.

4. All directory searches, master index evaluation, and granule pool evaluation is based upon F00 and later releases of BPM/BTM.

OPERATING SUGGESTIONS

Since any time-sharing activity is bound to cause additions and/or deletions from the granule pool maps, the VERIFY option will create situations where no results will be valid unless the NOMAP option is used. If the NOMAP option is used in this situation, FANALYZE will check other aspects (except the map) of the files found in the system directories.

Since FANALYZE must be in the :SYS account to run (due to the master mode CAL) and does not print its control card, it is suggested that FANALYZE be formed as a processor. Load particulars are as follows:

TSS	may be at least 200 (uses BPM stack).
ABS	may be set at the user's discretion.
LMN	may be any name (it does not check name).
SL	is 0.

Any other load option is incidental to operation.

ERROR PROCESSING

All snaps (hex dumps, error messages, etc.) produced during FANALYZE processing are preceded by a one-word title. These one-word titles are listed below.

ANLX-DMP All snaps with this title are due to granule map comparisons. The map words that failed comparisons are found in R2 and R3. The starting addresses may be found in R6 and R7. The index value to both maps is located in R1.

CALLSET Any snap with this title resulted from an I/O failure (i.e., a TYC of 8 or 9). The I/O calling sequence is placed into the registers as they were for the I/O call to NEWQ. The registers are as follows:

R0	End-action address (word address).
R1	End-action information.
R12	In byte format, FC/PRI/NRT/DCTX where FC is the I/O function code, PRI is the priority for this request, NRT is the number of recovery tries, and DCTX is the DCT1 index.
R13	Address of intended buffer (byte address).
R14	Read/write size (number of bytes).
R15	Disk address (DCTX/sector number).

An additional snap is'taken of the buffer that was due to receive the data (BUFFER). At this point, the program error exits.

FDKEY A snap with this title is a snap of the current file directory key that contained an invalid file information table address.

OLINKER A snap with this title is a snap of the current index used at a BLINK failure when the ANLZ option is being used. No repair is made or attempted.

CHAINERR A snap with this title is a snap of the current pyramid (upper level index granule) block on a BLINK/FLINK error. No repair is made or attempted. This error snap is encountered only when the ANLZ option is running.

PYR-ERR A snap with this title is a snap of the current upper level index granule on BLINK failures. This will occur while the ANLZ option is attempting to find the first block on the highest level and finds a BLINK failure.

CHN-ERR A snap with this title is a snap of the current index block on a BLINK/FLINK failure if the DATA option encounters a link failure.

FANALYZE error messages are listed in Table 5.

Table 5. FANALYZE Error Messages

Message	Description
ABOVE DATA ADDRESS BAD	An invalid master index data address location was found during the ANLZ run.
ACCOUNT DIRECTORY AND FILE DIRECTORY CONFLICT	The granule pool map showed dual allocation of granules for the two directories.
ACCOUNT DIRECTORY BAD, CANNOT RECONSTRUCT HGP	A link failure occurred in the account directory.

Table 5. FANALYZE Error Messages (cont.)

Message	Description
ACCOUNT DIRECTORY BAD, HGP RECONSTRUCTION HALTED	VERIFY found a bad ACNCFU address or a nonzero BLINK in the first AD sector.
ADDRESS xxxx NOT WITHIN PFA AREA	An address (xxxx) was found to be outside the PFA area.
BAD FIT ADDRESS	The file directory key contained on invalid FIT disk address. The address is printed to the right of the error message.
BAD FREE SECTOR POOL LINKAGE	The free sector pool had bad linkages.
BLINK ERROR	The backward link test failed on a new sector. The location and BLINK are printed beneath the message.
DATA GRANULE ALLOCATION ERROR	A data granule location has previously been allocated.
DISC I/O READ FAILURE	The TYC returned at end-action indicated a read error.
DUAL ALLOCATION	Anyerror other than could be defined occurred (e.g., FANALYZE could not accurately determine who owns the other granule).
DUAL ALLOCATION IN FREE SECTOR POOL	The granule allocation indicated the FSP entry conflicted with other allocation.
FILE CONFLICTS WITH ACCOUNT DIRECTORIES	The granule allocation for this file conflicted with the allocation for the account directory.
FILE CONFLICTS WITH FILE DIRECTORIES	The granule allocation for this file conflicted with the allocation for a file directory.
FILE CONFLICTS WITH PREVIOUS FILES	The granule allocation for this file conflicted with the allocation for other files.
FILENAME DOES NOT CORRESPOND TO FIT	The file directory entry name differed from the FIT name. The FD name is printed to the left of this message and the FIT is snapped.
FLINK ERROR	The forward address was invalid. The location and forward link are printed beneath this message.
INVALID CHARACTER OF OPTION	A bad option or comma appeared on a processor control command.
INVALID DATA ADDRESS	A key in a level 0 master index contained an invalid disk address. The address is printed to the right of the message.
INVALID FREE SECTOR POOL ENTRY	A bad address was found in the free sector pool. The address is printed to the right of the message.
I/O CALL FAILURE	A bad disk address has been passed to the disk I/O handler. The job aborts.
LINK ERROR IN ACCOUNT DIRECTORY	A BLINK/FLINK failure occurred in the account directory while the ANLZ option was running.
LINK ERROR IN FILE DIRECTORY	A BLINK/FLINK error occurred in the file directory while the ANLZ option was running.
MASTER INDEX ALLOCATION ERROR	A master index granule location has previously been allocated.

Table 5. FANALYZE Error Messages (cont.)

Message	Description
NAME SEQUENCE ERROR IN FILE DIRECTORY	The file directory names did not appear in ascending order. A dump is provided of that file directory sector.
NEXT LEVEL ADDRESS INVALID	The upper level index structure's next level address pointer was bad.
PYRAMID BLINK ERROR	A backward link (in a file's upper level index structure) has failed to verify.
PYRAMID FLINK ERROR	A forward link (in a file's upper level index structure) has failed to verify.
PYRAMID LOCATION ERROR	The TDA (from the file's FIT) had an invalid address.
RANDOM FILE ADDRESS ERROR	The FIT pointed to an invalid data address for a random file.
STACK SIZE INVALID	The load module FANALYZE was loaded without at least a TSS of 200.

10. HARDWARE REQUIREMENTS

INTRODUCTION

To arrive at a useful machine configuration to provide for the various types of users of BPM or BTM, the manager of each installation must evaluate his requirements and the level of service he desires. A reasonable selection of hardware can only be made with a good knowledge of its intended use, including the portions of computing time devoted to time-sharing, batch, and real-time users. Other requirements that must be evaluated include the number and usage profiles of on-line users, the size of programs, and the I/O characteristics.

It is impractical to list all combinations of equipment that would support BPM or BTM in some manner. However, a minimum equipment configuration for BPM and a typical configuration for BTM can be specified. Since there are considerable differences in requirements between standard BPM systems and a BTM system, the systems will be discussed independently of one another.

BPM REQUIREMENTS

A minimum hardware configuration for BPM is given in Table 6. A system with such a configuration might be dedicated to the use of one processor, say FORTRAN, or might be one that uses only a few processors and has a few users. Table 7 lists the recommended additions to the configuration. Figure 26 illustrates a minimum BPM system for Sigma 5-8 computers.

Note that any real-time requirements are preemptive and installation-dependent. Therefore, no allowance is given here for these demands on the computer. Users with real-time applications must add core to support real-time programs and may suffer batch throughput loss due to less CPU availability.

The BPM resident Monitor requires 8K words of core.

Table 6. Minimum BPM Hardware Configuration

Number of Units	Model Number	Description
		Central Processor
1	8201/8202	Sigma 5 Control Processing Unit
		Core Memory
1	8261/8262	Memory module (32K)
		Memory Protect
1	8214	Memory write protection feature

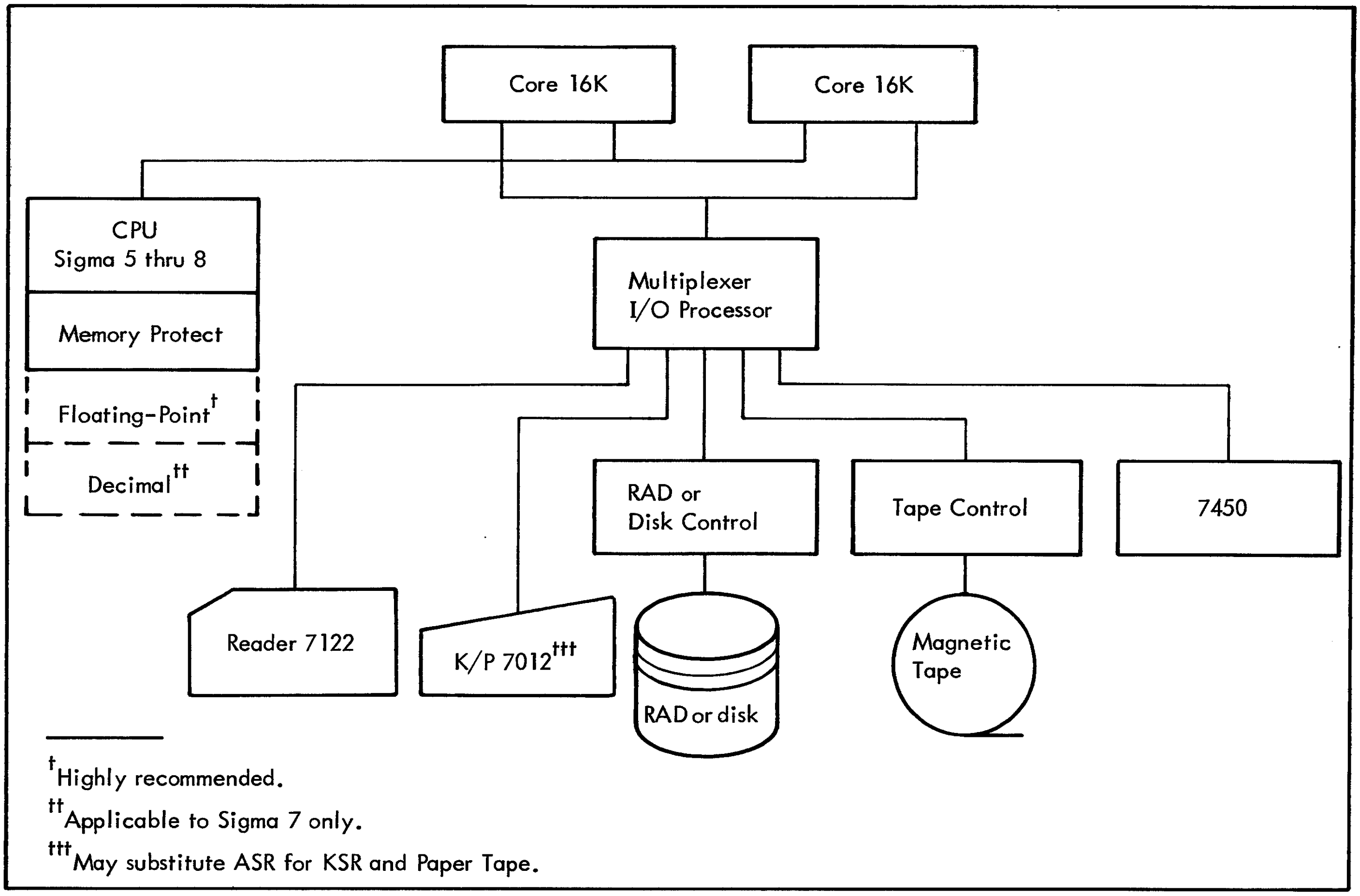
Table 6. Minimum BPM Hardware Configuration (cont.)

Number of Units	Model Number	Description
		Input/Output Processors
1	8203	Multiplexor IOP (except with 8201 CPU)
		Peripheral Equipment
1	7201 or 7240	RAD controller or Disk Pack Controller
1	7204 or 7246	RAD or Disk Pack Storage Unit (3 M bytes minimum)
1	7361	Magnetic Tape Controller (7-track)
1	7362	Magnetic Tape Unit (7-track)
1	7012	Keyboard Printer (may substitute ASR & paper tape)
1	7122	Card reader (400 CPM)
1	7450	Line Printer (low cost)

Table 7. Recommended BPM Hardware Additions

Number of Units	Model Number	Description
1	8252	Core Memory (8K)
1	8264	Additional Memory Port
1	8270	External Interface (Sigma 5 only)
1	8218	Floating Point (highly recommended)
1	7165	Card punch (100 CPM)
1	*7320	Magnetic Tape Controller (9-track)
1	*7322/7323	Magnetic Tape Unit (9-track)
1	*744X	Line Printer

*Recommended substitutes for units in Table 5.



[t] Highly recommended.

[tt] Applicable to Sigma 7 only.

[ttt] May substitute ASR for KSR and Paper Tape.

Figure 26. Sigma 5-8 Minimal BPM System

BTM REQUIREMENTS

The typical hardware requirements for a BTM system are given in Table 8. It is recommended that the 48K core memory be expanded to 56K if Meta-Symbol is used concurrently with on-line processing.

The amount of RAD storage is dependent upon the number of concurrent users and exact amount of storage per user. However, 3MB is required on permanent file RAD and/or disk pack for BTM, its processors, and libraries. In addition, 25-100K bytes per user is typical for file storage, assuming 24 users. Approximately 5.2MB on the swapping RAD is required for symbionts, on-line processors, and user swap area, assuming 24 users. For 38-user system, an entire 7212 is required for swapping. For a 64 user system, two 7212 RADs on one 7211 controller(HSRIOP for Sigma 8) are required. With two 7212 RADs, the on-line partition size may be enlarged but this will reduce capacity to less than 64 concurrent users. Swapping with 7242/7246 disk packs is allowable; response times, however, do not come near those predicted for various sized RAD systems. One RAD or disk pack controller is sufficient only for nonconcurrent operation.

A typical 48K BTM system is shown in Figure 27. (BTM supports a maximum of 128K words of memory.)

Table 8. Typical BTM Hardware Configuration

Number of Units	Model Number	Description
		Central Processor
1	8201 or 8202	Central Processing Unit
1	8218	Floating-Point Arithmetic
1	8221	Interrupt Control Chassis
1	8222	Priority Interrupts; 2-levels
1	8270	External Interface Feature (Sigma 5 only)
		Core Memory
2	8261/8262	Memory Module (48K)
1	8414	Memory Write Protection

Table 8. Typical BTM Hardware Configuration (cont.)

Number of Units	Model Number	Description
		Input/Output Processors
1	8203	Multiplexor IOP (required on Sigma 5 8202 CPU)
		Peripheral Equipment
1	7231/7241	RAD or Disk Pack Controller (1 Controller sufficient for non-concurrent operation only - see Figure 27)
1	7232/7242	RAD or Disk Pack Storage Unit (3 MB required on permanent RAD/disk pack file for BTM, processors and libraries)
1	7320	Magnetic Tape Controller (9-track)
1	7322/7323	Magnetic Tape Unit (9-track)
1	7012	Console Keyboard Printer (KSR)
1 1	7120 744X	Card Reader (400 CPM) Line Printer
		Communications Equipment
1	7611	Communications Controller
1	7612	Format Timing Groups
n[t]	7615	Formatted Send Modules (simplex transmit)
n[t]	7616	Formatted Receive Modules
n[t]	7620/7623	Interface Modules
n[t]	Various	Remote Terminals

[t] Number is equal to number of user terminals (up to 64).

REMOTE BATCH HARDWARE REQUIREMENTS

The hardware requirements for three types of remote batch configurations are shown in Figure 28. All data sets for remote batch must be supplied by the user.

Full duplex operation through the switched network is possible; however, no data set is yet available that will allow automatic establishment of the full duplex circuit. A full duplex circuit can be established through manual dialing and answering by operators at both ends of the circuit. In this case, XDS hardware requirements are identical to those shown in Figure 28 for full duplex operation over leased lines.

CORE MEMORY

Providing enough core memory is particularly crucial in a BTM time-sharing system. Operating with a memory of insufficient capacity for the load that is typical for an installation is especially detrimental to system performance (loader performance improves when more core is available than the absolute minimum).

Core space can also affect file applications; that is, if an average of five files are open concurrently, seven and one-half pages of core should be dedicated to file blocking. Less memory would result in system degradation.

BPM/BTM MONITOR SIZE ESTIMATION

The simple procedure given below, used in conjunction with Tables 9 and 10 and Figure 29 will give a general estimation of Monitor core size requirements, plus additional capabilities. More detailed and accurate formulas that are suitable for facility systems analysts are given in Appendix F. Note that the figures given in Tables 9 and 10 and Figure 29 represent the F01 version of BPM/BTM only, and are subject to change in any new versions of the system. The procedure is as follows:

1. Use Table 9 to obtain the size of the basic Monitor and add to it the size of any additional functional capabilities required (i.e., symbiont, remote batch, etc.).

2. Using Table 10, add the additional Monitor requirements for the desired peripheral configuration.

3. Take the sum of steps 1 and 2 and round it up to the next highest page.

4. The assumptions used to obtain these figures are listed in Figure 29.

5. The total size obtained from this procedure is that required for the Monitor itself, and does not consider

   - The size of the user batch partition.

   - The size of the on-line, time-sharing partition and context area.

   - The size of any resident or nonresident real-time requirements.

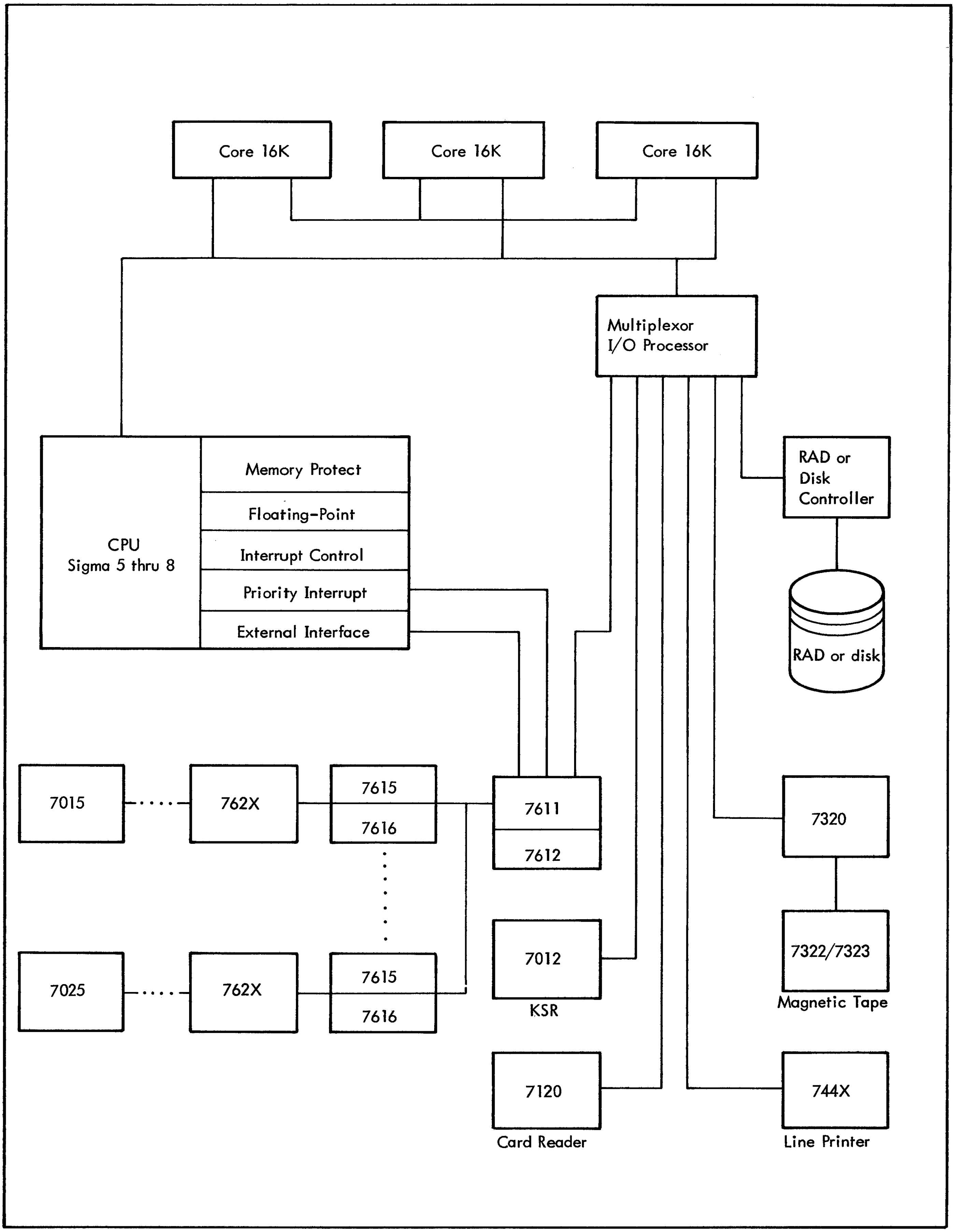


Figure 27. Typical BTM Hardware Configuration

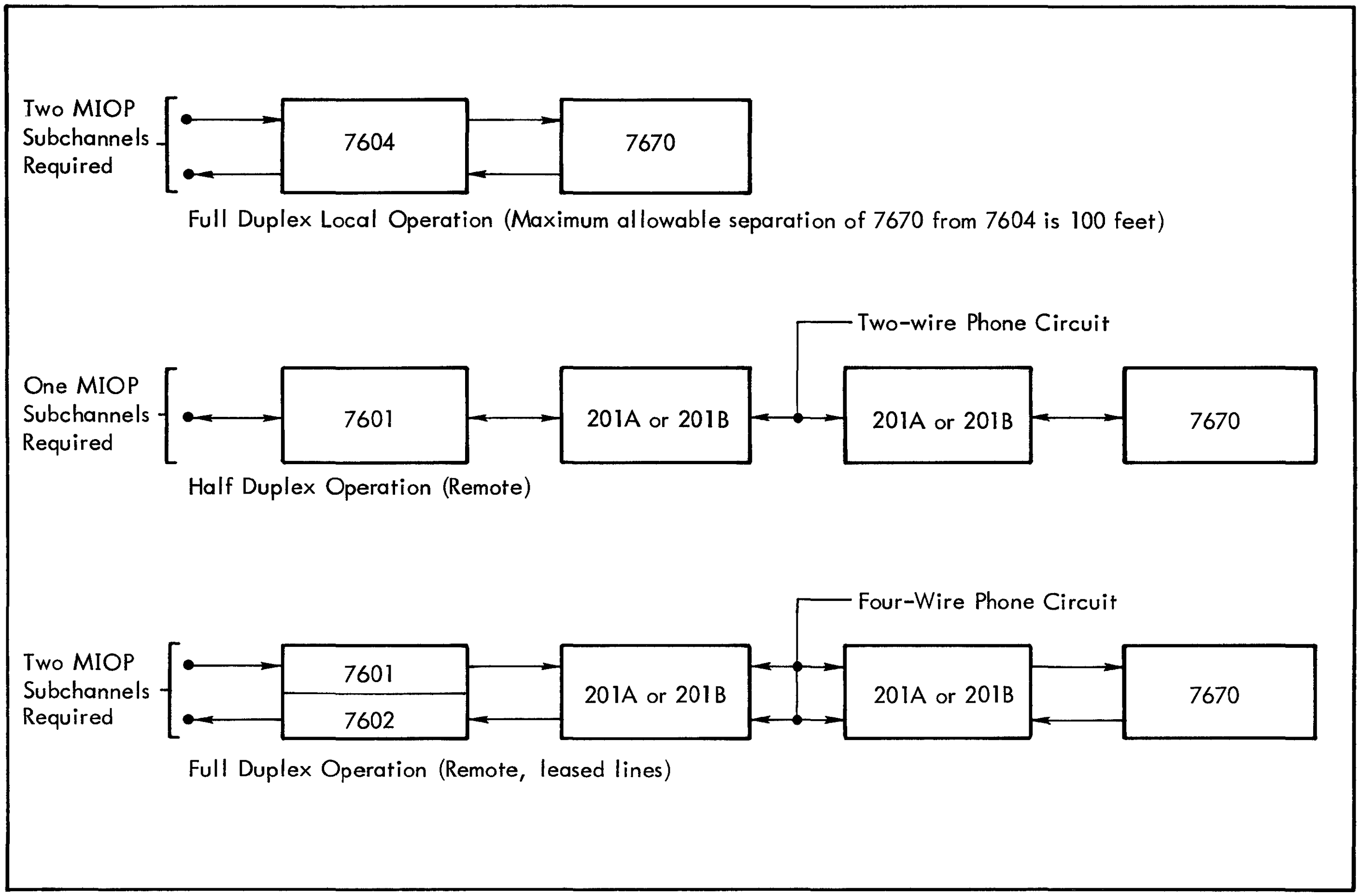

Two MIOP Subchannels Required
7604
7670
Full Duplex Local Operation (Maximum allowable separation of 7670 from 7604 is 100 feet)
Two-wire Phone Circuit
One MIOP Subchannels Required
7601
201A or 201B
201A or 201B
7670
Half Duplex Operation (Remote)
Four-Wire Phone Circuit
Two MIOP Subchannels Required
7601
7602
201A or 201B
201A or 201B
7670
Full Duplex Operation (Remote, leased lines)

Figure 28. Remote Batch Terminal Hardware Configurations

Table 9. Basic Monitor Size

Module	Core Required (Decimal Words)
BASIC MONITOR	8000
SYMBIONTS:	1200
First Card Reader	600
Additional Card Reader	300
First Line Printer (with C12)	600
Additional Line Printer	300
First Card Punch (with C12)	600
Additional Card Punch	300
REMOTE BATCH:	1200
Each 7601	300
BTM: (12 subsystems; 16K user size)	5000
Each User	110
BTM Performance Monitor	524

Table 9. Basic Monitor Size (cont.)

Module	Core Required (Decimal Words)
REAL TIME:	3000
Each External Interrupt (NINT)	12
Each R/T task (NFRGD)	12
Add for each Centrally Connected task (TSTACK)	50
File I/O buffer for FRGD	768

Table 10. Monitor Size Increase Decimal Words[†]

Model No.	First	Each Additional
7012	26	21
7020	26	21
7060	176	21
7121	28	23
7122	28	23
7140	28	23
7160	172	89
7165	109	89
7204	77	72
7212	116	111
7232	130	125
7242/6[††]	56	51
7242/6[†††] (handler included here)	544	413
7322 9[†]	31	26
7323 9[†]	31	26
7362 7[†]	31	26
7372 7[†]	31	26
7440 LP	26	21
7441 LP	26	21
7450 LP	26	21

[†] In the basic Monitor size, the following devices are already included: TY, CR, CP, LP, 7204, two 9-track magnetic tape devices. Therefore, these devices should not be counted again when using this table.

[††] Cylinder Allocated or Private.

[†††] Granule Allocated.

Basic Monitor:

TSTACK = 200 words

MPOOL = 4

IOQ = 10

CFU = 11

Symbionts: Buffering is computed for optimum efficiency

2 SPOOL + (1 CPOOL + 1 SPOOL)/SYMBIONT DEVICE TYPE

+ (1 CPOOL + 1 SPOOL)/SYMBIONT DEVICE

Remote Batch:

One CPOOL and one SPOOL buffer allowed for each 7601.

BTM:

User buffers	=	100-character input
		100-character output

1 CFU per user

Real Time:

Control task queue = 4

Figure 29. Assumptions

INPUT/OUTPUT PROCESSORS

The minimum requirement of one multiplexor I/O processor (MIOP) is reasonable for systems with up to 32 on-line users. For systems with more I/O devices (i.e., with more than 32 users) an additional MIOP may be required.

Bandwidth requirements of the IOPs must, of course, be met (see band width consumption in Figure 30). Heavy use of these devices may degrade system performance through CPU use and core buffer space required for I/O transfers. Generally speaking, additional core memory will improve performance in these large systems.

SECONDARY STORAGE

Requirements for RAD and/or disk pack storage may be divided into two categories:

1. Swap storage for on-line users, and for absolute core images of the BPM/BTM Monitor and system processors (FORTRAN, Basic, etc.).
2. File storage for all users of the system including those processors kept on file in load module (LM) form (usually SYSGEN processors, FORTRAN, COBOL, etc.)

Depending on user size and number of users, swap storage will usually require from one million to 12 million bytes on a 7212 RAD. It is necessary to have space to accommodate the aggregate size of all current on-line and multiple batch users. For instance a BTM configuration with a user size of 16K would require six megabytes of swap storage for 38 on-line users.

File storage is best estimated by the installation. For each on-line user, 25,000 to 50,000 words of file storage is frequently quoted as a typical need. Thus, if 100 users have access to a 32-line system, from 2.5 million to 5 million words of file storage would be needed, requiring two or three additional RADs (or disk packs). Table 11 gives the number of on-line users and percentage of batch usage for various BTM RAD and/or disk pack systems. All BTM configurations assume separate controllers for swapping and system devices.

In Table 11, the number of on-line users accommodated is based on "typical" conditions of on-line use and provides an "average" on-line response time of 2.5 seconds. These same numbers are usuable for nonconcurrent on-line or batch systems.

The % Batch Usage figures assume that time quanta are adjusted to provide a swap-limited system with the batch background consuming only swap time. Although the percentage of CPU time used for batch operations is independent of the system RAD, actual batch throughput will depend on the RAD used for system storage.

When used as a swapping file, the 7232 RAD may be equipped with a 7231 Controller and 7235 Extended Width Interface feature or a model 7236 Extended Width Controller. In both cases, the companion MIOP must be equipped with the 8X 75 Four-Byte Interface feature.

Note: When used for system files or user files, the 7231/7232 RAD system may not be equipped with a Model 7235 Extended Interface feature. A 7211/7212 RAD system requires an 8X 85 SIOP. The 7236 does not have this restriction.

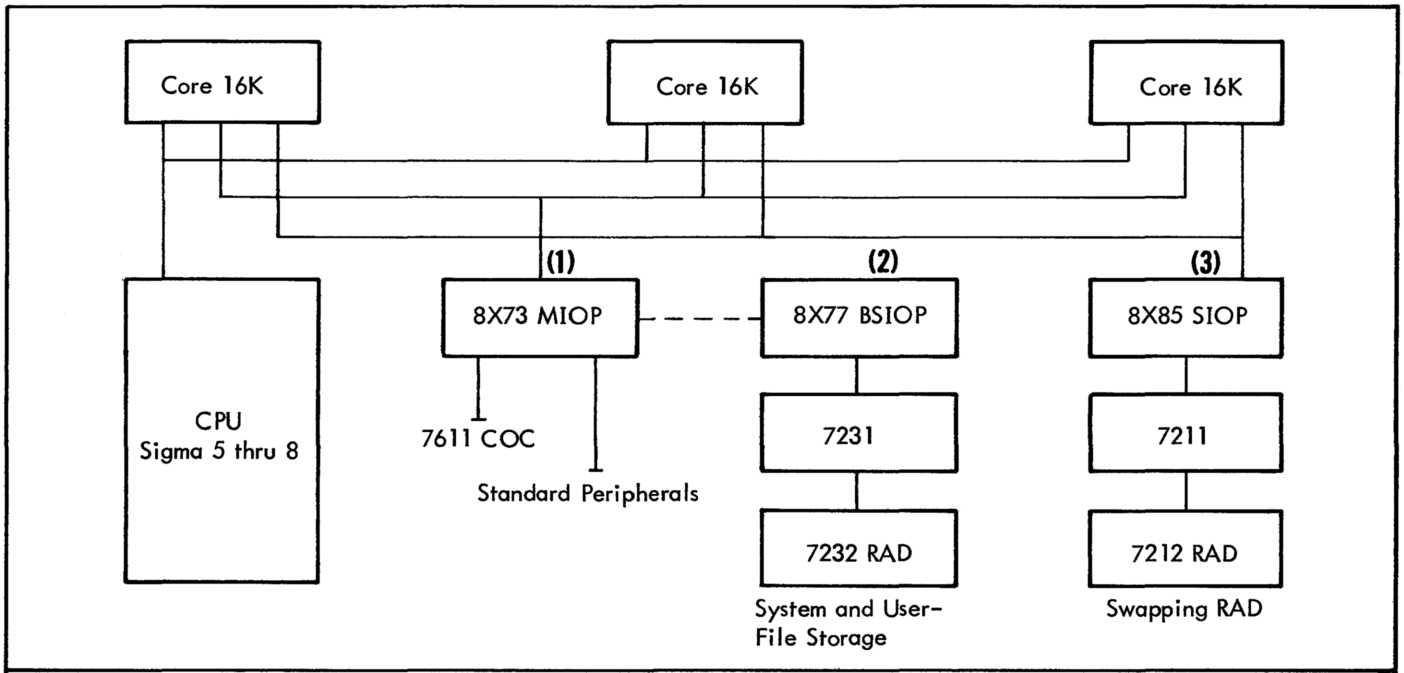
An example of a BTM RAD configuration is shown in Figure 30. Note that when configuring any BTM system, the MIOP bandwidths must be carefully observed.

Table 12 gives a sample of the contents of the file system :SYS account for a typical system. Individual installations will vary from this (considerably in some cases).

To maintain reasonable performance, separate RADs for swap and file storage are a minimum requiremenr for all systems. The 7212 high-speed RAD is recommended for swap storage and the 7232 RAD or disk pack is recommended for file storage.

Table 11. Typical Performance for Various Swapping Devices

Swapping Device	Number On-Line Users	% Batch Usage
7242/7246	8-10	80%
7204	16	60%
7232	32	35%
7212	40	20%



Notes:

(1)

8X73 Bandwidth Consumption:	%	Additional due to SIOP in System[†]	%
1 - 7122 Card Reader	1	-	1
1 - 7440 Line Printer	1	-	1
1 - 7160 Card Punch	11	2	13
1 - 7012 Keyboard/Printer	Negl.	-	Negl.
2 - 7322 80KB Tapes	16	2	18
- - 7611 COC	2	-	2
Bus-Sharing Effects[††]	5	-	5
	36		40

(2)

8X77 Bandwidth Consumption:	%	Additional due to SIOP in System[†]	%
1 - 7231/32 RAD	81	12	93
Bus-Sharing Effects[††]	5	-	5
	86		98

(3) This example represents a 38-user system. A second Model 7212 extends this configuration to the 64-user capacity.

[†] An HSRIOP is required for Sigma 8.

[††] An MIOP Channel B option is required for Sigma 8.

Figure 30. BTM RAD Configuration Example

Table 12. Typical Contents :SYS Account

File Name	Granules	File Name	Granules
:ACCTLG	†	ELIST	8
:BLIB	63	ERRDATA	2
:DIC	11	ERRMSG	3
:LIB	90	ERRWRT	2
:USERLG	††	FDP	39
BASIC	18	FERRET:	6[††††]
BASIC	20[††††]	FILEUP	17
BATCHACC	7[†††]	FLAG	63
		FMGE	8
BPM	10	FORTCOMP	8
BPM:	4[††††]	FORTLIB	5
BTMPM	44	FORTRAN	77
CCI	34	FORTRAN:	34
COBOL	228	FORTRANH	33
DEF	13	FPURGE	8
DEFCOM	5	IOTABLE	10
DELTA:	7	LOAD:	11[††††]
DICTNARY	12	LOADER	25
DMSDUMP	13	LOCCT	5
DMSINIT	10	LOPE	9
DMSLOAD	17	M:ABS	2
EDCON	8	M:AL	---
EDIT:	10	M:BI	---
M:BO	---	METASYM	61
M:BTM	10	MON	4
M:C	---	MONDUMP	42
M:CI	---	OLAY	24
M:CK	---	PASS1	15
M:CO	---	PASS2	37

Table 12. Typical Contents of :SYS Account (cont.)

File Name	Granules	File Name	Granules
M:CPU	20	PCL	13
M:DLIMIT	2	PFIL	9
M:DO	---	POST	3
M:EI	---	REPORT	10
M:EO	---	RETRIEVE	32
M:GO	---	REW	9
M:LI	---	ROMTRAN	14
M:LL	---	ROOT	3
M:MON	174	RUN:[††††]	6
M:OC	---	S:OVRLY	---
M:PO	---	SIGMET	5
M:SDEV	2	SIGFDP	5
M:SI	---	SL1	---
M:SL	---	SORT	18
M:SO	---	SSS	---
MACRSYM	26	SUPER	8
MANAGE	20	SUPER:	8[††††]
MCHKPT	3	SYMBOL	13
MEDDUMP	25	SYMBOL:	14[††††]
MERGE	9	VOLINIT	16
PASS3 11		WEOF	9

[†] Number of granules vary according to number of jobs run.

[††] Number of granules vary according to number of authorized users.

[†††] Accounting conversion from Version F01 back to E01 for Version E01 accounting users.

[††††] BTM version.

It is recommended that disk packs be included in the configuration. This will markedly improve the file storage capacity of the system. The disk pack equipment 7240, 7241, 7244 could be substituted for the 7231/7232 RAD devices. This also requires substituting the 8475 4-byte interface for the 8477 bus-sharing MIOP

PERIPHERAL EQUIPMENT

One on-site console keyboard printer is required for on-line monitoring and control of BPM/BTM. This is the operator's console. A card reader, line printer, and magnetic tape unit are also required minimum equipment. In addition, it is recommended that a second magnetic tape unit and card punch be added to the minimum equipment to provide a reasonable batch configuration.

TERMINALS

A variety of terminals may be used. This includes Xerox Keyboard/Printers (7015), Teletype Model 33 and Xerox Model 7670 Remote Batch Terminals. Software flexibility is present so that other terminals may be added with relatively little difficulty. One Model 7630 Communications Controller can handle up to 64 terminals. Up to 15 Remote Batch Terminals may be connected at any one time.

11. SYSTEM GENERATION OVERVIEW

INTRODUCTION

The process of generating an operating system is not simple. About 400 files must be manipulated and 400,000 to 500,000 I/O operations must be performed. Several processors are involved and a simple error occurring during the execution of any one of them may invalidate a system generation (SYSGEN) that took hours to perform. To the person who has never done a SYSGEN, the task seems formidable indeed. It need not be so. Anyone with the time, patience, and an understanding of what the SYSGEN process does should be able to generate his own system.

The purpose of this chapter is to give a general understanding of the system generation process by emphasizing the overall process rather than the detailed control command syntax and suggested values. A more detailed discussion is contained in the next chapter.

PURPOSE OF SYSGEN

An operating system is similar to other programs. It is assembled with Meta-Symbol and loaded with the loader just as a user program, say to calculate squares, would be. If that squares program could do its own I/O (or did not have any), it could be written on tape and booted into a machine just as the operating system is.

Everyone who might do a SYSGEN should be familiar with Meta-Symbol and the loader and be able to prepare the usual assortment of control cards, so why have this complicated procedure (and a new name-SYSGEN) for a common job? The answer is that this common job would be more difficult if it were done in the common way.

The loading process for a program (e.g., a program to calculate squares) is simple. A source deck is assembled into a relocatable object module (ROM) which is loaded into a load module and written on magnetic tape. The programmer need remember only the names of a few files.

Numerous source decks must be assembled into hundreds of ROMs during the SYSGEN process for an operating system. Certain of these, depending on the situation, must be loaded into the Monitor and processors. All of this must be put onto a magnetic tape in such a way that it can be read into the right places at boot time. The above process would alone require more of the programmer than is reasonable to expect. But this is not all that is required. Within the Monitor there are many tables that depend on the particular configuration on which the operating system is to run. In a program to calculate squares, a table might be changed with an assembly parameter; in the Monitor it would take hundreds of these to produce the correct tables.

Since programmers cannot (and certainly should not) be expected to remember all the necessary information to create an operating system, XDS provides the SYSGEN processors. The function of each of these processors is as follows:

Processor	Function
PASS1, PCL	Selects from various sources the relevant modules for system generation.
PASS2	Compiles the required dynamic tables for the resident Monitor.
LOCCT and PASS3	Store and execute load card images (by calling the loader) to produce load modules (LMs) for the Monitor and its processors.
DEF	Writes a Monitor system tape that may be booted and used.

It is the purpose of these processors to make the creation of a specific operating system as simple as possible. The SYSGEN process can appear complex if the programmer does not remember what it is that SYSGEN is trying to help him do, and that is to assemble, load, and write to tape a common, although large and complex, machine language program.

The purpose of this chapter is to help the reader understand the SYSGEN processors. All of the steps in the SYSGEN process will be covered at a level that should neither leave the reader without critical detail nor bog him down in detail. To the seasoned SYSGEN programmer, this chapter may seem over simplified. To the programmer who is undertaking his first SYSGEN, however, this chapter should make the SYSGEN process much easier and more successful.

GATHERING ROMs (USING PCL)

It is assumed throughout this section and the sections that follow that file space is not a problem. Of course, this is not always true. The steps that are necessary when file space is a problem are discussed in the last section of the chapter.

It is generally not necessary to assemble most modules used to build an operating system. This is done at XDS and the resulting ROMs are referred to as BI (binary input) or BO (binary output). However, some elements may have been updated and may require assembly before SYSGEN is begun. These should be kept in some account (cards, tape, or another account) other than the account in which SYSGEN is being done. Also, some elements may have been added by the user. These, too, should be kept in an account other than the SYSGEN account.

At this point, the user is ready to build the base for his SYSGEN. The first step, the assembly, has been completed and the next step is the load. This step consists of placing the desired element files (ROMs) in the target account. What is desired is a copy of every ROM needed to load the Monitor and all processors to be included in the final system.

The ideal target account for this stage of system generation is one that is empty at the beginning of the process. System generations can be done in any account but much trouble can be avoided if :SYSGEN is used as the target account.

Gathering ROMs consists of the following three steps:

1. COPYING ALL OF A PO TAPE INTO THE TARGET ACCOUNT.

   If this is the first system generation, there may be no PO tape. If not, this step should be skipped. A PO tape from an old release is better than none at all, but if this tape is not current it is better to proceed as if there were none.

   This step copies load modules instead of ROMs. The purpose of the step is to ensure that a copy of all processors, including those that are not affected by System Generation, is obtained. Since the last function of a SYSGEN process is to write a PO tape, this step ensures that everything necessary is included in the account at the start. The step may result in the gathering of items that are not wanted but these are overwritten later in the SYSGEN process.

   The easiest way to copy the PO tape into the target account is to use the PCL COPYALL command. This command would be written as follows:

   ```
   !PCL
   COPYALL LT#TAPE[.account] TO DC
   ```

   where TAPE is the INSN of the PO tape, and "account" is the account (usually :SYSGEN) under which the PO tape is created.

2. COPYING ALL OF A BI TAPE INTO THE TARGET ACCOUNT.

   This step gathers the standard ROMs supplied by XDS. Normally, the generation of a system will involve updating only a few ROMs. The rest of the required ROMs will be standard ROMs. The BI tape used in this step can be the master BI (not PO) tape used to boot the system, another BI tape, or a separate BI (ROM) tape. One of these tapes is supplied with each release of the system.

   The easiest way to copy the BI tape is to use the COPYALL command. The format of the command is

   ```
   COPYALL LT#TAPE [.account] TO DC
   ```

   where TAPE is the INSN of the BI tape.

   When this step is complete, the user is ready to go to PASS2, LOCCT, PASS3, and DEF to build the finished system, providing he does not want to update any of the ROMs on the BI tape. If he does, he must go to step 3.

3. COPYING UPDATED ROMS OVER THOSE ON THE BI TAPE.

   If the user has ROMs to update, he should by now have the assemblies done and have the updated ROMs in another account or on cards or tape. The next step in the process is to replace old ROMs.

   ```
   COPY CR OVER DC/ROM1
   <BINARY DECK FROM UPDATED ROM1>
   !EOD
   !EOD
   COPY CR OVER DC/ROM2
   <BINARY DECK FOR UPDATED ROM2>
   !EOD
   !EOD
   COPY LT#UPDT/ROM3 OVER ROM3
   COPY LT#UPDT/ROM4 OVER ROM4
   COPY ROM5. :UPDATE OVER ROM5
   COPY ROM6. :UPDATE OVER ROM6
   ```

   In this example, the updated copies of ROM1 and ROM2 are binary decks, those of ROM3 and ROM4 are on tape #UPDT, and those of ROM5 and ROM6 are files in account :UPDATE. Any of these could be new ROMs rather than updated ones.

   This completes the gathering of ROMs for the SYSGEN. If the latest version of all ROMs is to be saved for a future SYSGEN, DEF can be used to write a BI/BO tape with a bootable system and a ROM portion like the one used in step 2.

   ```
   !ASSIGN M:BO,(DEVICE,9T),(SN,XBO1)
   !DEF BPM[, versn#]
   :WRITE BO
   END
   ```

   where versn# is a 3-character version number (e.g. F01).

BUILDING DYNAMIC TABLES (PASS2)

For a program, such as one to calculate squares, steps 1 through 3 could be replaced with one assembly and building dynamic tables for the Monitor would not be necessary. However, for an operating system, these steps are necessary and the dynamic tables must be built for the Monitor.

Dynamic Monitor tables depend on the physical devices present on the target machine, the amount of file space, the number of BTM users, and a variety of other installation-dependent parameters. This data cannot be generated at XDS since it is different for each user and each use of the Monitor. It would be too much to ask each user to assemble his own tables, so the PASS2 processor is provided to build them. The output of PASS2 is a set of library load modules, but these are used, like ROMs, as elements in loading the Monitor, and thus should be thought of as binary output.

The discussion below is not concerned with command syntax since that is covered in the next chapter. It is instead a discussion of what commands to use, the order of those commands, the meaning of the commands, and the pitfalls to avoid.

REQUIRED COMMANDS

The following is a list of control commands that should be used in every BPM or BPM/BTM PASS2 process:

```
:CHAN
:DEVICE
:MONITOR
:DLIMIT
:ABS
```

If the system is a symbiont system, the following command is necessary:

```
:SDEVICE
```

If it is a real-time system, PASS2 requires

```
:FRGD
```

If it is a BTM system, PASS2 requires

```
:BTM
```

If the system does not have a card reader, line printer, or card punch, the following command is required:

```
:STDLB
```

COMMAND SEQUENCE

PASS2 allows the commands to be ordered in many ways. However, the following command sequence will always work:

```
:CHAN/:DEVICE
[:STDLB]
[:SDEVICE]
:MONITOR
:DLIMIT
:ABS
[:FRGD   ]
[[:INTLB]]
[:BTM]
```

MEANING OF PASS2 COMMANDS

The discussion below briefly outlines the function of each of the PASS2 commands.

:CHAN establishes a logical channel for all of the devices between it and the next :CHAN. It should always be the first control command in the set of commands for PASS2. In general, every device except a tape, RAD, or disk pack should have its own :CHAN. All RADs, disk packs, and tapes that have the same nd of yyndd (e.g., 9TA80) should be on the same channel.

```
Wrong               Right

:CHAN               :CHAN
:DEVICE TYA01       :DEVICE TYA01
:CHAN               :CHAN
:DEVICE LPA02       :DEVICE LPA02
:DEVICE CRA03       :CHAN
:CHAN               :DEVICE CRA03
:DEVICE 9TA80       :CHAN
:CHAN               :DEVICE 9TA80
:DEVICE 9TA81       :DEVICE 9TA81
:DEVICE 9TAF1       :CHAN
:DEVICE 9TB82       :DEVICE 9TAF1
                    :CHAN
                    :DEVICE 9TB82
```

:DEVICE defines a device attached to the machine for which the system is being generated. In general, the only option necessary is name (e.g., TYA01). If the device is the COC for BTM, the option HANDLERS must be specified. For example,

```
:DEVICE COA10,(HANDLERS,COC,COC)
```

RADs and disk packs should have a type (7212, 7232, etc.) specified and one or more of the following: PSA, PER, ABSF, BCHK, and PFA. Other disk options may apply to a particular installation. If the device is not standard, all options should be specified.

:STDLB defines the standard operational labels. If the computer does not have a card reader, line printer, or card punch, the labels assigned to these devices should be assigned to an existing device or to NO, e.g., :STDLB (EO,NO). Output assigned to NO is discarded. Even if the computer has the devices listed above, it may be desirable to change some of the default op label assignments with :STDLB.

Two files are written as a result of the :CHAN, :DEVICE, and :STDLB commands. They are IOTABLE, which contains tables for the I/O system, and SPEC:HAND, which is a list of handler names used by PASS3.

:SDEVICE specifies which devices are symbiont and generates M:SDEV, which contains symbiont tables.

:MONITOR specifies system parameters and generates M:CPU, which contains buffers and system tables.

:DLIMIT respecifies the system's default limits for jobs to be processed. If the standard default limits are acceptable, this command need not contain any options. :DLIMIT generates M:DLIMIT, which is loaded into CCI.

:ABS defines the ABS (absolute) area on disk and names processors to be made ABS processors. Making a processor absolute shortens the time necessary to load it into core. The M:ABS module is generated as a result of this command. If there are no ABS processors, the command should read

```
:ABS, 1024
```

where 1024 is the minimum required to load the Monitor. In this case, a message, '('EXPECTED BUT NOT FOUND, is printed but should be ignored since it is a warning that the user may have forgotten to include processor names.

:FRGD specifies the system's real-time parameters and generates M:FRGD, which contains buffers and tables for real-time operations.

:INTLB allows the user to label interrupt locations. These definitions are placed in M:FRGD.

:BTM specifies parameters concerning the BTM area of the system and generates the M:BTM module with BTM tables and buffers.

PASS2 PITFALLS

If PASS2 is run as described above, no error messages should be output. A list of control commands not used will be printed at the end of PASS2. This list should be checked but need not be a cause of concern. A very lengthy PASS2 takes less than five minutes, so there is no reason to take chances with error messages. If any error messages appear, the appropriate corrections should be made and the entire PASS2 should be run again. This can be done repeatedly at this stage of SYSGEN without disturbing the BO base that was created and should be done until PASS2 is run without errors.

PASS2 cannot determine whether the parameters that have been specified are reasonable for the system so long as they are within legal limits. Therefore, it is important that the user read the next chapter and make a careful selection of PASS2 values.

PASS2 should never be run unless the exact computer configuration is known. It is critical that the devices that are defined with :CHAN and :DEVICE command exist and are at the addresses specified. The exact configuration of the computer can be obtained from the XDS field analyst.

LOCCT FILES

At this point, every element file necessary to load the Monitor and all of the processors should be in the target account. The next step is to load the system. To load the Monitor (M:MON), PASS3 must be used instead of the loader. For ease and consistency, PASS3 may be used to load the entire system. PASS3 requires LOCCT files.

DEFINITION OF LOCCT FILES

The loader does not read control cards. !LOAD (!OVERLAY, !OLAY) and !TREE commands contained in a job deck are read by CCI which uses them to construct a control table for the loader. This table is written to the ABS area on the RAD and the loader is called by CCI. Upon entry, the loader reads in the table and uses it to do its loading.

PASS3 also calls the Loader and therefore must be able to supply a control table. This is done with LOCCT files. When CCI reads a !LOCCT command, it processes it as though it were a !LOAD writing the same table to the RAD, but the LOCCT processor is called instead of the loader. The LOCCT processor reads a command specifying a name (1 to 10 characters) and opens a file whose name is the name on the card with the five letters LOCCT appended to the left. LOCCT then reads the control table that CCI built from the RAD and writes it to that file. The file is then called a LOCCT file. For example,

```
!LOCCT (LMN, TEST), (BIAS, O), (MAP), ;
!(SL, F), (PERM), (NOTCB), (NOSYSLIB), ;
!(EF, (A), (B), (C), (D))
!TREE A-(C, D)-B
!DATA
:LOCCT TESTX
```

This builds the file LOCCTTESTX which contains the control table for loading TEST from element files A, B, C, and D.

GENERATING LOCCT FILES

When a system with a standard Monitor and standard processors is being built, the necessary LOCCT files are contained on the BI tape and are in the target account at this point. These files are listed and their contents are described in the documentation supplied with the system release. If the standard Monitor and processors are adequate, the LOCCT pass is not necessary and PASS3 can be run. However, if any of the following is true, LOCCT files must be built:

1. The LOCCT file for something to be loaded is not on the BI tape.
2. Modules are to be added to or deleted from the Monitor (or processor).
3. A load option or the overlay structure of the Monitor or a processor is to be changed.
4. A nonstandard XDS product is to be loaded with PASS3.
5. The target account used for SYSGEN is not :SYSGEN. (LOCCT files must be built in the account in which the

SYSGEN is done, and the LOCCT files on the XDS BI tapes are built in the :SYSGEN account. If the target account is not :SYSGEN, a complete new set of LOCCT files must be built. The ones supplied by XDS can be used as models.)

Building LOCCT files is not a complicated process. The procedure to create LOCCT files is as follows:

1. Make out !LOAD and !TREE commands as though the load module were going to be loaded directly by the loader.

2. Make a !DATA card and one that says

   !LOCCT name

   where "name" is the name of this LOCCT file.

3. Punch LOCCT instead of LOAD into the !LOAD card.

4. Arrange the deck in the following order:

```
!LOCCT
!TREE
!DATA
:LOCCT
```

If a new BO tape is to be written during the gathering of ROMs, the new LOCCT files should be created just before writing that tape so that they will be included and be ready for the next SYSGEN.

LOADING THE MONITOR AND PROCESSORS (PASS3)

The next step is to load the Monitor and processors. If PASS3 is used, it must have the LOCCT files described in the previous section. When PASS3 reads the command :TESTX, it reads the file LOCCTTESTX and writes the control table it contains to the ABS area on the RAD just as CCI does when it processes !LOAD commands. It then calls the loader which reads in the table and uses it to do the load. The following two command sequences are identical except that A produces the additional file LOCCTXYZ:

A

```
!LOCCT (LMN,XYZ),(EF,(A),(B),(C),(D))
!TREE A-(C,D)-B
!DATA
:LOCCT XYZ
!PASS3
:XYZ
```

B

```
!LOAD (LMN,XYZ),(EF,(A),(B),(C),(D))
!TREE A-(C,D)-B
```

If PASS3 reads a LOCCT file in which the load module name (LMN) is M:MON, it takes the following action:

1. It reads in the file SPEC:HAND which was generated in PASS2 and obtains from it the names of the device handlers needed in the system. It then reads in the ROMs (always getting BASHANDLES) for all the needed handlers and combines them into one file named HANDLERS.

2. It generates the file ROOT which contains information necessary to manage the overlay structure.

3. It records the Monitor's background lower limit and uses it as the default bias when loading processors.

The system can now be loaded.

ITEMS TO BE LOADED

If there were no changes to be made to the Monitor either with code updates or with PASS2 parameter changes there would be no reason for doing a SYSGEN. Therefore, M:MON must be loaded. It should be loaded first so that the rest of the processors will be biased correctly. In general, CCI should also be loaded to use the new limits generated by :DLIMIT in PASS2.

If this is the first SYSGEN of a new release, all of the processors should be loaded. An example of this PASS3 load operation is included with the release documentation.

If this is not the first SYSGEN for a release, copies of the processor load modules are already in the target account. These were brought in from the PO tape at the time the ROMs were gathered. Therefore, only those processors that have been updated need be loaded. However, if it is not known which processors have been updated, it is better to load too many than not enough. If the background lower limit of the Monitor has changed, all the processors should be loaded to provide correct biasing.

OPTIONS TO BE USED

On the !PASS3 card, the option MON should be specified. This will cause PASS3 to abort if the Monitor load is unsuccessful. The !PASS3 card for BPM should read

```
!PASS3 BPM,MON
```

The rest of the options appear on PASS3 control commands and should be used as follows:

1. DELETE and SAVE should not be ûsed since file space is assumed not to be a problem (until the last section of this chapter).

2. BIAS = VALUE should be used only if a processor is to be biased at a specific location other than the background lower limit.

3. BIAS = +offset should be used only if a processor (e.g., a BTM subsystem) is to be biased a certain amount above the background lower limit.

The majority of the processors should be biased at the background lower limit and therefore BIAS should not be specified for them.

ERROR PROCEDURES

PASS3 runs can produce two different sets of error messages: those that are preceded by asterisks and come from PASS3, and those that are preceded by a number of the form 30200XX and come from the loader. Most of the PASS3 messages are self-explanatory (Chapter 12 describes how to correct these errors). There are a few PASS3 error messages that require special action. They are listed below.

1. ***OPEN M:EI ERR/ABN = 0003(LOCCT)

   This message appears when the file LOCCT name does not exist ("name" is the name on a PASS3 control command). The name should be checked to ensure that the name specified is the name of the LOCCT file rather than the name of the load module to be built.

   ```
   Right                     Wrong

   !LOCCT (LMN,M:MON)        !LOCCT (LMN,M:MON)
   !DATA                     !DATA
   :LOCCT TMON               :LOCCT TMON
   !PASS3                    !PASS3
   :TMON                     :M:MON
   ```

   Then M:EI should be checked to ensure that it is assigned to FILE in the target account. PASS3 reads the LOCCT files through M:EI. If this DCB was assigned earlier in the job, PASS3 may be searching in the wrong place.

   If the user has not found the problem at this point, the LOCCT file does not exist. It should be built as specified above (see the section titled "LOCCT Files").

2. OTHER ERROR MESSAGES REFERRING TO LOCCT, BIN, CARD OR CHECKSUM

   These messages should be treated with caution because similar messages are printed by the loader. If any of these are printed by PASS3, the LOCCT file is bad. It should be rebuilt as described above (see the section titled "LOCCT Files").

3. ***OPEN/READ BASHANDL FILE ERR/ABN = xxxx

   or

   ***OPEN/READ SPEC:HAND FILE ERR/ABN = xxxx

   These messages should appear only when M:MON is being loaded. Although PASS3 will continue, the Monitor will not be properly built. If the first message appears, BASHANDL should be reassembled and the ROM should be put into the user's SYSGEN account. If the second message appears, PASS2 must be run again.

Loader errors are described in Appendix C of the BPM/BP,RT Reference Manual, 90 09 54. Refer to that manual for the cause of the error.

PASS3 must be rerun only on those loads that have errors. The fact that some modules did not load correctly does not mean that the others are not correct. Exceptions are

1. If M:MON does not load correctly, the entire PASS3 run should be rerun.

2. If anything must be reassembled (the ROM was bad or didn't exist), all modules that use this ROM as an element should be reloaded.

3. If PASS2 is run again, M:MON (and therefore everything else) must be reloaded.

WRITING PO TAPE (DEF)

At this stage, a completed system is in the target account. If the system is a real-time system, it may be necessary to add some programs to the tape about to be written so they will reside in the :SYS account when the tape is booted. These programs should be copied to the target account in load module form at this point.

The PO tape is written in the following way:

```
!ASSIGN M:PO,(DEVICE,9T),(SN[xxxx )]
!DEF BPM[versn#]
:INCLUDE (:BLIB,SIG7FDP,BPM)
:WRITE PO
END
```

where xxxx is the serial number for the tape and versn# is a 3-character version number (e.g., F01). The INCLUDE card is not necessary if METASYM, LOPE, and the on-line loader are not in the system.

HELPFUL HINTS

- HAVE ALL ASSEMBLIES DONE BEFORE YOU BEGIN SYSGEN

  This just good organization. There is no reason to have to worry about compressed and update files at SYSGEN time. The best approach is to do all of the new assemblies early and to put the updated ROMs on a tape. The entire BO base can then be obtained by the following:

  ```
  !PCL
  COPYALL LT# PO TO DC
  COPYALL LT# BO TO DC
  COPYALL LT# UPDT TO DC
  END
  ```

where LT# PO is a PO tape, LT# BO is a BO tape, and LT# UPDT is the user's tape of updated and new ROMs.

- DIVIDE SYSGEN INTO AS MANY SEPARATE JOBS AS POSSIBLE

Nothing is more frustrating than to submit an entire SYSGEN as one job and to wait hours to find out that a minor error was made on one of the first commands. When the SYSGEN is divided into many jobs, the output of each stage can be checked for errors before the next stage is started. The most efficient job setup is probably as follows:

```
!JOB
!PCL
DELETEALL
COPYALL LT# PO TO DC
!JOB
!PCL
COPYALL LT# BO TO DC
!JOB
!PCL
COPYALL LT# UPDT TO DC
LIST DC
```

At this point, it is possible to determine whether everything was copied correctly, whether the ROMs that were expected are present, and what LOCCT files are present.

```
!JOB
!PASS2
:CHAN
   :
END
```

PASS2 can be checked here to see if it ran correctly. If not, it should be rerun.

```
!JOB    }
!LOCCT  }
!TREE   }   This is required only if LOCCT
!DATA   }   files must be built.
:LOCCT  }
!LOCCT  }
!DATA   }
:LOCCT  }
   :    }
```

The LOCCT files should then be checked to see if they were built properly.

```
!JOB
!PASS3 BPM,MON
:<LOCCT NAME FOR M:MON>
:<LOCCT NAME FOR CCI>
END
```

The Monitor and CCI can now be checked to see whether they were loaded correctly. If not, they should be loaded again.

```
!JOB
:PASS3
(Load the rest of the desired items.)
END
```

After this step, the loaded items should be checked. If any item did not load, it should be corrected and reloaded.

```
!JOB
!ASSIGN M:PO,(DEVICE,9T),(OUTSN,XPO)
!DEF BPM,F01
:INCLUDE (:BLIB,BPM,SIG7FDP)
:WRITE PO
END
```

If the DEF worked, the SYSGEN is complete. If not, corrections should be made and the DEF run again.

- USE THE FOLLOWING PROCEDURE

The SYSGEN procedure below is written in the form of a program. It includes all of the steps necessary for the successful generation of an operating system.

1. If you have no modules to update or add, go to step 4.
2. Assemble the modules you wish to update or add.
3. Copy the new ROMs to a tape.
4. If this is the first SYSGEN for a release, go to step 7.
5. COPYALL a PO tape into :SYSGEN.
6. If step 5 was not successful, go to step 5.
7. COPYALL a BI tape into :SYSGEN.
8. If step 7 was not successful, go to step 7.
9. If you do not have an update tape from step 3, go to step 12.
10. COPYALL your update tape into :SYSGEN.
11. If step 10 was not successful, go to step 10.
12. Run PASS2.
13. If step 12 was not successful, go to step 12.
14. LIST DC.
15. If you want to change any LOCCTs, go to step 17.
16. If LOCCTs for everything you need are in :SYSGEN, go to step 19.
17. Build any LOCCTs that are necessary.
18. Go to step 16.

19. If you wish to write a new BO tape, do so.
20. If step 19 was not successful, go to step 19.
21. Run PASS3 to load M:MON and CCI.
22. If step 21 was successful, go to step 25.
23. If M:MON did not load, go to step 21.
24. If CCI did not load, reload it.
25. If this is not the first SYSGEN for a release, go to step 28.
26. Run PASS3 to load every processor.
27. Go to step 29.
28. Run PASS2 to load the processors you have updated.
29. If any processor load was unsuccessful, repeat it.
30. Write your PO tape.
31. If step 30 was unsuccessful or if you want more PO tapes, go to step 30.
32. Stop.

LIMITED FILE SPACE

In the previous sections of this chapter, it has been assumed that plenty of file space exists for SYSGEN. This is not always true. This section describes what to do when there is not plenty of file space. It assumes a knowledge of the information contained in the previous sections.

The approach of using two PCL COPYALL commands to gather the binary output from lengthy tapes is obviously not valid here. If all the binary output needed is obtained at one time, there may be more files than available space. Therefore, it is necessary to gather just enough ROMs to do one load and to delete them before going to the next load. At the end of this process, all of the needed load modules will be in the target account, and all unnecessary ROMs will have been deleted. This SYSGEN will have more steps and take longer, but it is basically just as simple as any other.

GETTING DESIRED FILES

There are two ways to get certain files from the BO tape. If desired, these files can be copied one by one using PCL.

```
!PCL
COPY LT# BO/FILE TO DC/FILE
COPY LT# BO/FILE2 TO DC/FILE2
```

where FILE and FILE2 are the names of the desired files. The Monitor and some of the processors have a great many element files, however, and copying them file by file is a big effort. For this reason, XDS BI/BO tapes have standard files. These files are lists of the ROMs needed to load a particular Monitor or processor and the necessary LOCCT file. A standard file and all the files listed in it can be copied from the tape using PASS1.

```
!ASSIGN M:BI,(SN,BO),(LABEL,X)
!PASS1
:SELECT (STD,$::BPM)
```

where $::BPM is the name of the standard file and BO is the SN of the BI tape. If none of the standard files lists all of the desired files, the following command may be added to the commands above:

```
:SELECT (FILE,A,B,C)
```

The single files, as well as those in the standard files, will be copied. Remember that standard files and LOCCTs go together and that if no standard file is desired, a new LOCCT will probably have to be built.

DELETING FILES THAT HAVE BEEN USED

The DELETE parameter on PASS3 control commands causes all the element files used in the load to be deleted. If this parameter is used and only the desired files are obtained from the BI/BO tape, only the desired load module will remain in the target account after PASS3 (plus whatever was there before). For example,

```
!PASS3 BPM,MON
:TEST (DELETE)
```

However, if PASS3 makes explicit reference to a ROM in some other account, DELETE will attempt to delete that ROM from the account under which PASS3 is currently running.

SPECIAL CONSIDERATIONS

The procedure described above makes the handling of new and updated ROMs more complicated. After the ROMs for the Monitor or a processor have been obtained, only the relevant update files must be copied. This can also be done with PASS1.

```
!ASSIGN M:EI (SN,UPDT),(LABEL,X)
!ASSIGN M:BI (SN,BO),(LABEL,X)
!PASS1
:SELECT (STD,$::BPM)
:UPDATE (FILE,A,B,C)
```

where A, B, and C are updated ROMs to replace those in $::BPM.

The Monitor should be loaded first, followed by CCI, to use and delete the modules built by PASS2. After that, the desired processors should be loaded beginning with the largest and proceeding in descending size. The reason for this is that the space available for ROMs decreases as load modules that have been loaded accumulate.

Since the PO tape was not copied, there are no copies of processors not updated. These should be copied file by file from a PO tape after all of the PASS3 runs have been completed. If ROMs such as :BLIB, BPM, and SIG7FDP from the PO tape are desired, they should be copied at this time. Do not forget to load or copy the system DCBs.

Briefly, here are the steps that you should follow:

1. Get the ROMs necessary to load the Monitor via PASS1 or PCL (this includes any updated or new ROMs).
2. Run PASS2.
3. Build a LOCCT if necessary.
4. Run PASS3 with DELETE to load the Monitor.
5. Get the ROMs for CCI.
6. Build a LOCCT if necessary.
7. Load CCI (with PASS3 and DELETE).
8. Start with the largest processor you want to build.
9. Get the ROMs for this processor.
10. Build a LOCCT if necessary.
11. Load the processor (with PASS3 and DELETE).
12. If you have loaded all of the updated processors, go to step 15.
13. Choose the next largest of the processors you want to load.
14. Go to step 9.
15. Copy the processors you have not loaded (including system DCBs) from a PO tape.
16. Copy :BLIB, SIG7FDP, and BPM if desired.
17. Write your new PO tape.
18. Stop.

12. SYSTEM GENERATION DETAILS

PARAMETERS DEFINING TARGET SYSTEM

Before attempting to generate a system for a specific target installation, the user must know certain things about that system. He must know the desired characteristics of the system so that he can incorporate this data in the appropriate System Generation control commands.

To define the characteristics of the target system, the user should determine the answers to the following questions:

1. Is the default set of Monitor operational labels, and the standard assignments for those labels, adequate for the target system? If not, what standard labels and standard assignments are required?

2. What peripheral devices are needed and which of these should share the same channel controller? Are special I/O optimization routines to be used in the target system?

3. What I/O handlers are to be used in the target system? If special handlers are needed, what are the names of the primary and secondary entry locations?

4. How many tracks are available on the RAD or disk pack to be used in the target system? How many sectors are there on each track, and how many words per sector? Which tracks, if any, may not be used by the target system? How many RAD or disk pack tracks are needed for symbiont queue storage, permanent file storage, permanent system storage, checkpoint storage, and absolute foreground programs?

5. Are symbionts to be used in the target system? If so, what devices are to be used for symbiont I/O?

6. How many jobs may be present at any one time in symbiont input and output job queues?

7. How many words of core storage are needed for the Monitor's temporary storage stack?

8. How large is the core memory to be used by the target system?

9. How many I/O operations may be queued at any one time?

10. How many files may be open at any one time?

11. How many DCBs may be open at any one time?

12. What is the address of the first unused, even-numbered interrupt available to the target system for the origin of the generated Monitor?

13. How many words of core storage should be reserved for patching the Monitor?

14. Are the standard defaults for LIMIT parameters adequate for the target system? If not, what default limits should be established?

15. What processors, if any, are to be entered into the target system in absolute format? How many words of RAD or disk pack storage are needed for the storage of absolute processors?

16. How many foreground programs, if any, are to be included in the target system?

17. Which interrupt location is to be used by the Monitor's control task, and how many CALs may be queued by the control task at any one time?

18. How many pages of core storage are needed for foreground common reference (i.e., foreground COMMON area)?

19. How many pages of core storage are needed for resident foreground programs?

20. How many buffers should be pooled for foreground file indexing, and how many should be pooled for packing and unpacking foreground data?

21. How many interrupts may be in use at any one time?

22. Which foreground programs, if any, are to be absolutized, how many words of RAD or disk pack storage should be allocated to provide room for program expansion, and how many pages of core storage should be allocated for each when loaded into core for execution?

23. Which interrupt locations, if any, may be referred to in Monitor CALs by symbolic labels? What labels will correspond to these locations?

24. What user programs, processors, or other program elements are to be established as standard systems? What load structures are to be defined for these standard systems, and what names are to be used in referencing their LOCCT tables?

25. In addition to any standard system files, what other files are to be included in the target system? Are all such files present on the current master system tape, or must some be obtained from an update tape or deck?

26. What patches, if any, must be made to the Monitor or to system files after the target system has been booted from the generated system tape?

SYSGEN PROCESSORS

BPM System Generation is a multi-pass process by which the user can generate a BPM operating system tailored to the requirements of a specific installation. Starting with a BPM master system tape, the user can create a bootable system tape from which the generated BPM operating system can be loaded into a target machine. The target machine can be any Sigma system having a hardware configuration compatible with BPM (see "Hardware Requirements", Chapter 10). One Sigma Computer (Sigma 5-8) can be used to generate a BPM for another Sigma computer, or vice versa, and the target machine may have more or less core storage than the one used to generate the system tape.

The master system tape contains a bootable Monitor, files of load modules comprising the processors and other routines to be used during System Generation, and a large number of element files that constitute a data base for the System Generation process. The user may patch the operating system as it is loaded into the machine from tape but cannot alter the object modules at this time, since the object modules are not read from the tape until the PASS1 processor is called. When the Monitor has been booted and the nonresident routines have been written to the disk, the BPM system is fully operational.

System Generation control commands read by the PASS1 processor allow the user to select element files from the data base of the master system tape, to substitute updated files for these (if necessary), and to add files to the resulting revised data base which is maintained in disk storage for use in later phases of the current System Generation. The DEF processor provides the option of writing a revised master system tape for use in some future System Generation. The PASS2 processor reads System Generation control commands and generates disk files of load modules that establish operational labels, peripheral characteristics, real-time interrupts, and other installation-dependent parameters for generated system tape written during a later phase of the current System Generation. PASS2 may be performed either before or after PASS1, since the two are entirely independent.

The object modules selected during PASS1 must be combined in load module form before a generated system tape can be written. Also, the tree structures for any overlays must be established. A tree table for each BPM standard system is present in the master system tape. If the user references a standard system (e.g., Meta-Symbol) in a control command during PASS1, he need not make explicit reference to the tree table; it will be included automatically. However, tree tables for nonstandard systems must be created by the user through appropriate !LOAD and !TREE control commands.

After the user has created a tree table for an overlay structure, he has the option of calling the LOCCT processor to generate a permanent LOCCT file containing the tree information, so that this information need not be created anew during subsequent System Generations.

If the generated system is to include BPM standard systems or user-defined programs having associated LOCCT files of tree information, the PASS3 processor must be called to initiate the formation of load modules for such systems or programs.

The PASS3 processor reads control commands specifying which LOCCT tables are to be used to define the load structure of BPM standard systems or user-defined programs. The user has the option of altering the load bias of a defined load structure and may specify that a given LOCCT table and associated object modules are to be deleted from disk storage when the component object modules have been loaded.

The first command read by PASS3 should specify the Monitor's LOCCT table (e.g., LOCCTBPM57M), so that the Monitor will be loaded first. This will enable PASS3 to establish minimum bias values for each subsequent load structure. If a user-specified load bias value is less than the minimum set by PASS3, it will be used although a warning message will be output on the LL device. If the Monitor is not the first item loaded, any items loaded prior to the Monitor will not be biased automatically but will be biased according to the value given in the pertinent PASS3 control command or, if no such value was given, according to the bias contained in the LOCCT table for the item.

Items not specified in PASS3 control commands may be loaded via !LOAD, !OVERLAY, or !OLAY and !TREE commands as in ordinary batch processing, except for the Monitor load module (M:MON). M:MON must be loaded via PASS3.

When all desired object modules have been converted to load module form, the DEF processor must be called to write a tape containing the generated system. The !DEF command may be followed by :INCLUDE, :IGNORE, :WRITE and/or :DELETE control commands. The :INCLUDE command specifies the names of any user files (e.g., data, object modules, procedures, etc.) to be included in the generated system tape although not in load module format. The :DELETE command specifies that all object modules in the current user's account are to be deleted from disk storage; otherwise, they will be saved for use in future System Generations. The :IGNORE command specifies the load modules to be excluded from the system.

The ~~system tape generated by the~~ DEF processor has the same general format as the master tape (Figure 31) used in booting the BPM system employed in the System Generation process. The method of loading the generated system into the target machine is identical to that used in booting from the master tape.

Detailed procedures for generating standard (i.e., typical) BPM systems, and detailed descriptions of the various control commands used in System Generation are presented later in this chapter.

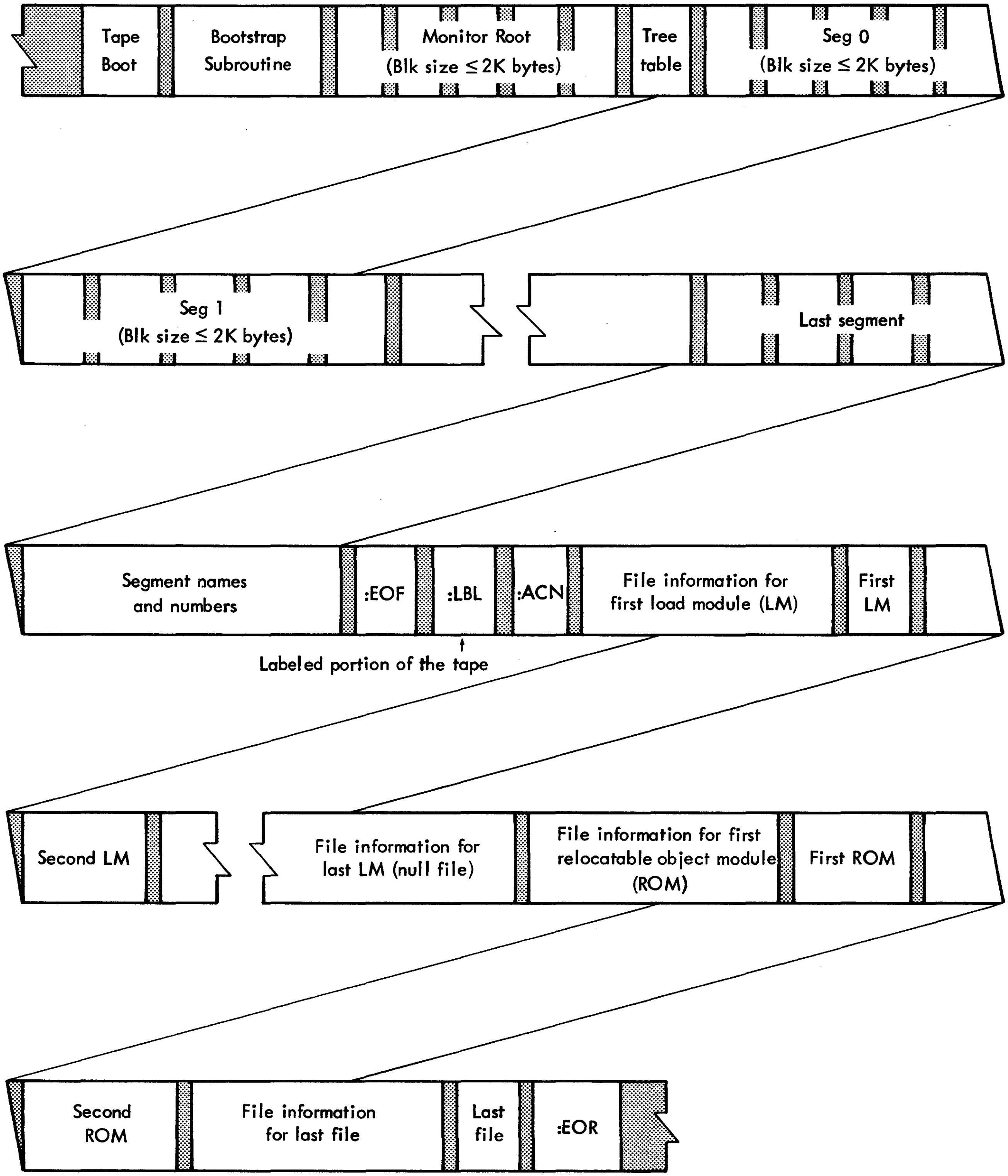

Tape Boot
Bootstrap Subroutine
Monitor Root (Blk size ≤ 2K bytes)
Tree table
Seg 0 (Blk size ≤ 2K bytes)
Seg 1 (Blk size ≤ 2K bytes)
Last segment
Segment names and numbers
:EOF
:LBL
:ACN
File information for first load module (LM)
First LM
Labeled portion of the tape
Second LM
File information for last LM (null file)
File information for first relocatable object module (ROM)
First ROM
Second ROM
File information for last file
Last file
:EOR

Figure 31. Format of Master System Tape

COMMAND FORMAT

The formats and functions of the control commands used in the various phases of System Generation are discussed below. The control commands used in System Generation are of two general types: Monitor control commands having an "!" in column 1 (e.g., commands used to call System Generation processors for execution) and System Generation control commands having a ":" in column 1 (i.e., commands used to communicate optional or required parameters to System Generation processors).

Since the user has considerable flexibility in setting up and performing a System Generation, it is not practical to present exhaustive examples of System Generation deck setups in this manual. However, by observing the general considerations outlined in this chapter, the user should have little difficulty in setting up his own System Generations.

Many users will find that the "cookbook" job setups given for generating standard BPM systems are adequate for their needs, and it is likely that the requirements of most installations can be met by relatively minor alterations of one of the standard configurations.

PASS1 PROCESSOR

The PASS1 processor allows the user to select (and/or update) and write to disk all nonkeyed element files required for later phases of System Generation or for the creation of a new master system tape.

The PASS1 processor accepts selected files from the BI device and updates files from the EI device. The default assignment for M:BI is to the BI device and for M:EI is to the card reader. If the user prefers to use other media for these functions, he must assign M:BI and/or M:EI appropriately before calling PASS1.

The output tape DCB is opened with a DEVICE assignment to prevent problems caused by a LABEL assignment.

PASS1 recognizes the following commands.

```
:SELECT     :LABEL

:UPDATE
```

!PASS1 This command causes the Monitor to fetch the PASS1 processor from the RAD disk pack and to transfer control to it.

The !PASS1 control command has the form

```
!PASS1 option
```

where option specifies the type of Monitor the user wishes to generate. BPM, BTM, and UTM are legal types. The default type is that of the Monitor that is running.

Once PASS1 assumes control of the system, it continues until a control command beginning with an exclamation character (except an !EOD command) is read.

:SELECT This command selects specified files and/or standard system files from the BI device. Selected files are retained on the RAD or disk pack for use in subsequent passes of System Generation. More than one :SELECT command may be used. The :SELECT command has the form

```
:SELECT (option)[,(option)]...
```

where the options are

ALL specifies that all files read from the BI device are to be selected. If ALL is specified, no other options are needed.

FILE, file name,... specifies the name of an element file that may be used with a processor, library, or Monitor configuration. More than one such name may be specified in a single FILE option, but no file name may exceed 15 alphanumeric characters. Disk files of another account can be selected and updated by assigning the DCB to a file name and account.

STD, standard system file name specifies the name of a file that contains, as individual records, the file names of the elements comprising a specific processor, library, or Monitor configuration. Each record in a standard system file contains an element file name (similar to Symbol/Meta-Symbol TEXTC format) beginning with the first byte of the record, (e.g., 8PASS1ROM). There may be no more than one standard system file name per STD option, but there may be more than one STD option in a :SELECT command.

The PASS1 processor reads (from the BI device) the element file names contained in each selected standard system file and subsequently selects the corresponding element files (also from the BI device). Since PASS1 reads the master tape only once, each selected standard system file must precede any of its referenced element files. Because files on the master tape are arranged in alphanumeric order, standard system file names must precede any of their element file names in the alphanumeric hierarchy (e.g., standard file BETA must not contain the name of element file ALPHA, because file BETA would not be read from the tape until after element file ALPHA had been passed and ignored).

To select files from a device other than the master tape read by PASS1, the user must assign M:BI appropriately.

:UPDATE This command replaces (or inserts) specified files on the RAD or disk pack with element files from the EI device. If update files are input from the card reader (the default assignment for M:EI) each file deck must be preceded by a :LABEL command (see below) and followed by an !EOD command.

The :UPDATE command has the form

```
:UPDATE (option)[, (option)]
```

where the options have the same form as for the :SELECT command (see above), but update files are always read from the EI device rather than the BI device. Update files are always obtained after the select file phase is completed. :UPDATE commands are honored even if there are no selects.

:LABEL This command specifies the name of the data deck that follows the :LABEL command. The deck sequence must be arranged as shown in Figures 32 and 33. Note that the first use of a :LABEL command must be preceded by a !EOD card.

The :LABEL command has the form

```
:LABEL, name
```

where name specifies the name of the element file or standard system file contained in the data deck that follows. The name may not exceed 15 alphanumeric characters in length.

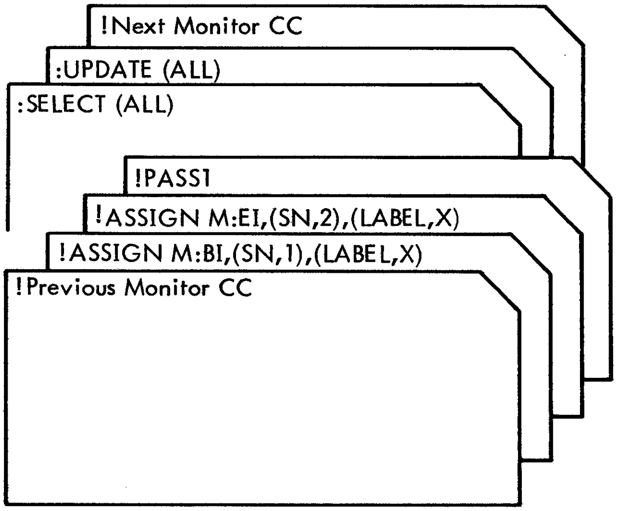
A :LABEL command must be used for each file that is to be selected and/or updated from the card reader.

PASS1 MESSAGES

Table 13 lists the messages that are output during PASS1 processing. All messages are output on the LL device.

PASS1 EXAMPLES

Examples of typical PASS1 deck setups are shown below.



```
!Next Monitor CC
:UPDATE (ALL)
:SELECT (ALL)
!PASS1
!ASSIGN M:EI,(SN,2),(LABEL,X)
!ASSIGN M:BI,(SN,1),(LABEL,X)
!Previous Monitor CC
```

This deck setup would cause PASS1 to select every file from the BI device (labeled tape) and then select every update file from the EI device (labeled tape), replacing or supplementing the elements originally selected from BI.

CREATION OF ERRMSG FILE

After PASS1 has been processed, the ERRMSG file should be created using the ERROM program. This program runs as a batch job with data records read via the M:SI DCB.

Data Formats:

First Field	(Cols. 1-6)	:	Hexadecimal Code
Second Field	(Cols. 7-80)	:	Text of Message

Example:

```
020022 NO ROOM TO READ LIBRARY REF/DEF STACK
```

The last two digits of the 6-digit code are the error code.

The BI tape contains two files, ERROM and ERRDATA. ERROM is the ROM of the program and ERRDATA is the sequential file of data records. After performing a PASS1, the following jobs should be run:

```
!JOB :SYS, ME, F
!LOAD (EF, (ERROM, :SYSGEN))
!ASSIGN M:SI, (FILE, ERRDATA, :SYSGEN)
!RUN
!JOB :SYSGEN, ME, F
!ASSIGN M:EI, (FILE, ERRMSG, :SYS)
!ASSIGN M:EO, (FILE, ERRMSG)
!FMGE (ENTER, PERM)
```

These two jobs ensure that the error message file exists in the :SYS account for the duration of the SYSGEN and that it also exists in the :SYSGEN account in order that DEF will write it to the PO tape as a keyed file.

PASS2 PROCESSOR

The PASS2 processor may be used either before or after the execution of PASS1 to define the foreground and background environment of the target system. PASS2 reads control commands from the C device and generates files of load modules on the system RAD or disk pack.

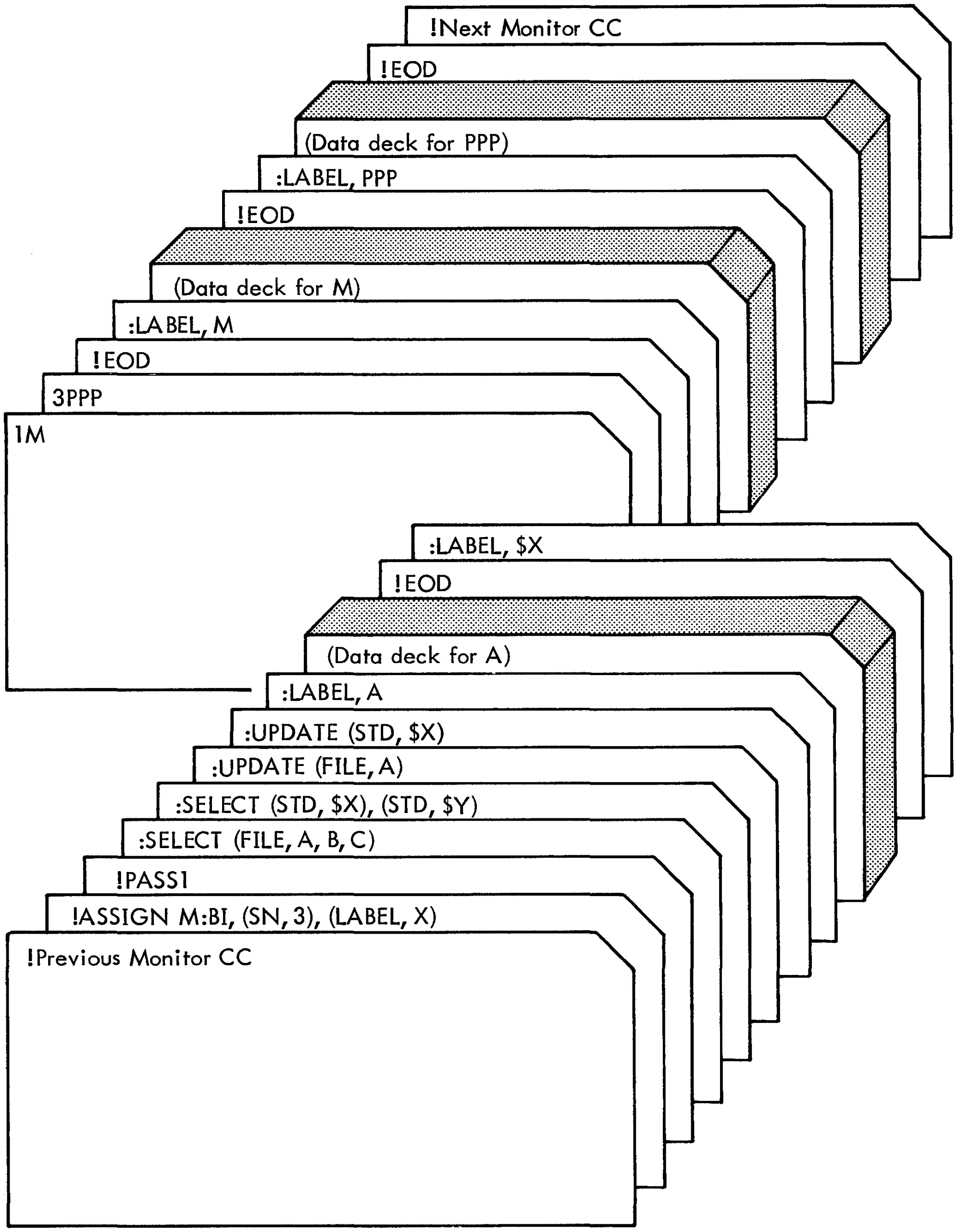
The following are SYSGEN PASS2 size restrictions when defining peripheral devices and standard labels:

64 unique type mnemonics

128 STDLB control commands

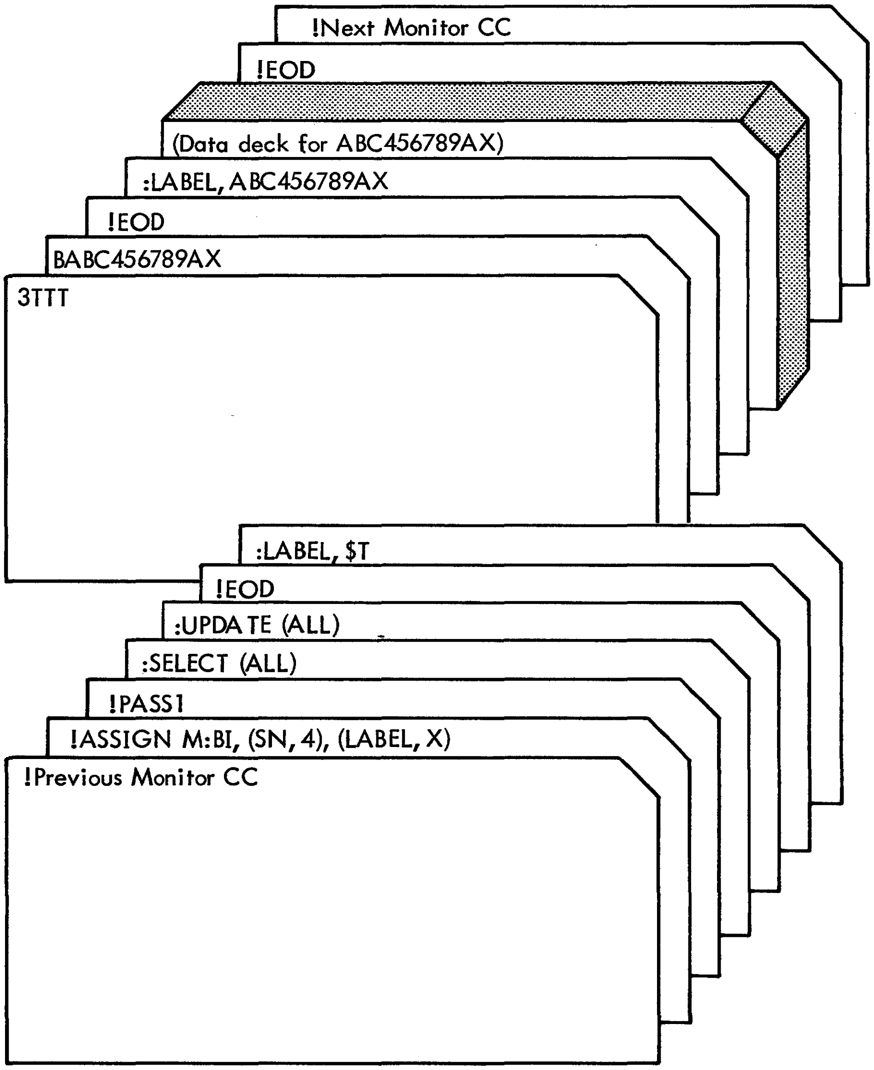
32 CHAN control commands

96 DEVICE control commands



The deck setup would cause PASS1 to select files A, B, and C and standard-system files $X and $Y from the BI device (labeled tape). Standard-system files $X and $Y will contain the names of other files to be selected from BI. Update file A and standard system file $X will be obtained from the EI device (card reader). Notice that the standard system file $X precedes the files named in its own records.

Figure 32. PASS1 STD Files



This deck setup would cause PASS1 to select every file from the BI device (labeled tape) and then to select every update file from the EI device (card reader) replacing or supplementing the elements originally selected from BI. Notice that the update file ST is a standard system file. Its records contain the file names that make up standard file $T. However, in this case, the update file $T is not looked at as a standard system file, but as just a normal update file. File $T can be used in future System Generations as a standard system file.

Figure 33. PASS1 Use of SELECT/UPDATE ALL

Table 13. PASS1 Messages

Message	Description
n ABNORMAL ON xxxx	An I/O abnormal condition has been encountered on the specified device. The abnormal code is nn; the name of the DCB for the device is xxxx. If the DCB is M:BI or M:EI, PASS1 continues; otherwise, it aborts.
****BO TAPE CONTENTS****	The names following this message are the names of all files output on the BO tape. PASS1 continues.
CONTROL COMMAND ERR	The PASS1 command encountered a control command that was not recognized (including a LABEL card before an !EOD), contained a syntax error, or had no colon in column 1. PASS1 makes an error return to the Monitor.
DELIM ERR	PASS1 encountered a control command containing an incorrect delimiter. PASS1 makes an error return to the Monitor.
FILE NOT ON UPDATE CARD–IGNORED	A file name on a LABEL card did not appear as a file name on an UPDATE card. PASS1 ignores the corresponding file from the M:EI device and continues.
NAME ERR	PASS1 encountered a control command containing an illegal name (i.e., one having no alphabetic character or one having a nonalphanumeric character). PASS1 makes an error return to the Monitor.
NAME FROM BI/EI STD FILE ERR	PASS1 encountered a file name in one of the records of a standard file that is greater than 15 characters in length. PASS1 makes an error return to the Monitor.
NO BO WILL BE GENERATED	This message is always output during PASS1. PASS1 continues since this is not an error condition.
****SELECT FILES NOT FOUND****	The names of files specified by SELECT command were not found on the BI device. (The names follow the message.) PASS1 continues.
****STD FILES NOT FOUND****	The names of files specified in STD options were not found on the BI device. (The names follow the message.) PASS1 continues.
***UNKNOWN TYPE-xxx USED	The Monitor type option was not present or legal. "xxx" is replaced with the default type (of the Monitor that is running).
****UPDATE FILES NOT FOUND****	The names of files specified by an UPDATE command were not found on the EI device. (The names follow the message.) PASS1 continues.

32 unique handlers (i.e., unique devices with handler names)

32 unique operational labels

32 RADs or disk packs defined by DEVICE control commands

The following commands are recognized by PASS2, and continuation cards are allowed.

:STDLB

:CHAN

:DEVICE

:DLIMIT

:ABS

:FRGD

:SDEVICE

:MONITOR

:BTM

:INTLB

!PASS2 This command causes the Monitor to fetch the PASS2 processor from the RAD or disk pack and to transfer control to it.

The PASS2 command has the form

```
!PASS2 option
```

where option specifies the type of Monitor the user wishes to generate. BPM, BTM, and UTM are legal types. The default type is that of the Monitor that is running.

When PASS2 assumes control, it continues until a control command with an ! in column 1 is encountered.

:STDLB This command defines standard Monitor operational labels for the target system. If used, it must appear with the :CHAN/:DEVICE commands. It may appear either before or after a :CHAN or :DEVICE command and more than one :STDLB command may be used.

The :STDLB command has the form

```
:STDLB (label,name)[,(...)]...
```

where

label specifies a Monitor operational label comprising 1 or 2 alphanumeric characters, at least one of which must be alphabetic.

name specifies a physical device name (to which the label, above, is to be assigned) or an operational label. Device names have the form yyndd (see Tables 14, 15, and 16).

Table 14. I/O Device Type Codes

Device (yy)	Physical Device Name
MT	Magnetic tape
7T	7-track magnetic tape
9T	9-track magnetic tape
CP	Card punch
CR	Card reader
PP	Paper tape punch
PR	Paper tape reader
TY	Typewriter
LP	Line printer
DC	Magnetic disk
DP	Disk Pack
PL	Plotter[†]
NO	No device
CO	COC[†]

[†] Handlers must be specified.

Table 15. Channel Designation Codes

Specified Channel Letter(n)	Corresponding Decimal Digit of Unit Address
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7

Table 16. Device Designation Codes

Hexadecimal Code (dd)	Device Designation
00 ≤ dd ≤ 7F	Refers to a device number (00 through 7F)
80 ≤ dd ≤ FF	Refers to a device controller number (8 through F) followed by a device number (0 through F).

If no :STDLB commands are used, the following default set of labels is assumed; if one or more :STDLB commands are used, then none of these defaults is assumed.

C = LI = SI = BI = CI = EI = device CRA03

OC = device TYA01

LO = LL = DO = SL = device LPA02

PO = BO = SO = CO = AL = EO = device CPA04

:CHAN This command groups peripheral devices (see :DEVICE below) according to channel controller. All :DEVICE commands following a given :CHAN command must be a part of that channel. At least one :CHAN command must be used, and each :CHAN command must precede the :DEVICE command (or commands) to which it applies.

The :CHAN command has the form

```
:CHAN
```

:DEVICE This command specifies the name and characteristics of a system peripheral device. One :DEVICE command must be used for each device in the target system. Any combination of secondary storage devices (up to 32 devices) can be specified.

The :DEVICE command has the form

```
:DEVICE name[, (option)]...
```

where name specifies the device name (see Table 14, 15, and 16) and the options are as follows:

{INPUT / OUTPUT / IO} specifies whether the device is to be used for input, output, or both. The default is IO.

HANDLER, name1, name2 specifies the name of the I/O handler to be used. Name 1 is the primary entry (build command list and start device) and name 2 is the secondary entry (handle interrupt). Each name must not exceed 7 alphanumeric characters. If this option is omitted, the default handler for the device type is assumed (see table below). Name 1 must be the name of the object module for this particular handler unless it is one of the defaults.

Device Type	Name1	Name2
TY	KBTIO	KBTCU
PR	PTAP	PTAPCU
PP	PTAP	PTAPCU
CR	CRDIN	CRDINCU
CP	CRDOUT	CRDOCU
LP	PRTOUT	PRTCU
DC	DISCIO	DISCCU
9T, 7T, MT	MTAP	MTAPCU
DP	DPAK	DPAKCU

The above names must be used unless the user has supplied his own handler as an object module.

PAPER, size, width specifies the (hexadecimal) number of printable lines per page (size) and the maximum (hexadecimal) number of characters per line (width). This option applies to typewriters, Teletypes, and line printers. If this option is omitted, the values 38_{10} and 132_{10} are assumed for size and width, respectively.

The allocation of disk area may be constrained by the following options. All defaults are 0 unless otherwise indicated.

type specifies, by device model number, the values indicated below for SS, NSPT, and SIZE. The default type for DC is 7204 and for DP is 7242.

RAD/Disk Pack	SIZE_{16}	NSPT_{16}	SS_{16}
7204	200	10	5A
7212	40	52	100
7232	200	C	100
7242	FA0	6	100

CYLINDER specifies that a cylinder allocation table is to be built instead of a granule allocation table and is used for disk packs only. A cylinder allocated device can be either public or private. For DP allocation, there are 400 allocatable cylinders (30 512-word granules or 1/2 a physical cylinder).

Cylinder allocation requires that the options SIZE, PFA, PSA, BCHK, and ABSF for disk devices be specified in hexadecimal cylinders, instead of hexadecimal tracks. The PER option is ignored for cylinder allocation. SIZE is automatically changed to 400_{10} if it was 4000_{10}.

If cylinder is not specified, BPM will assume granule allocation.

PRIVATE specifies that the file will be recognized as private and CYLINDER allocation is assumed. The PRIV bit is set in the allocation table, the PUB bit is reset in the Automatic Volume Recognition (AVR) table, and PFA is set equal to SIZE, ignoring all other options (PER, PSA, ABSF, BCHK, and PFA).

PFA, value specifies, in hexadecimal, the number of tracks (cylinders) to be allocated for permanent file storage, including element files. The default for DC devices is 130_{16}.

PSA, value specifies, in hexadecimal, the number of tracks (cylinders) to be allocated for permanent storage on DC or DP devices. The default on DC devices is 40_{16}. This read contains the absolute core image of the Monitor, Monitor overlays, and absolutized processors (see "ABSF"). It also contains the absolute secondary storage read/write area and one sector for each of the 36 standard DCBs.

BCHK, value specifies, in hexadecimal, the number of tracks (cylinders) to be allocated for background checkpoint storage. This area must be large enough to hold the entire background when a nonresident foreground program which is biased in the background is to be executed, a resident foreground program is loaded into the background via an M:ABSLOAD call, or an M:SBACK call is issued by a real-time program. The default is 0.

ABSF, value specifies, in hexadecimal, the number of tracks (cylinders) to be allocated for absolute foreground programs formed by specifying the ABS option in the !RUN command or with the INTS option in the :FRGD command for SYSGEN. This space must be large enough to hold all of the absolute foreground programs. The default is 0.

PER, value specifies, in hexadecimal, the number of tracks to be allocated for peripheral symbiont queue storage. The default value is 90_{16} for DC devices (note that one minute of backup for an 800 line/minute printer uses $25,000_{10}$ words of disk scratch storage). The PER option is ignored for cylinder allocation.

The Device Control Tables (DCT) will be reordered according to the device type to eliminate order requirements for the AVR table, and the granule pools (HGP) will be reordered to ensure that disk devices with permanent system storage (PSA) precede others.

The following options further define the files for disk devices:

NSPT, value specifies, in hexadecimal, the number of disk sectors per track.

SS, value specifies, in hexadecimal, the number of words per disk sector.

SIZE, value specifies, in hexadecimal, the number of tracks (cylinders) available to the system on this device.

SIZE, SS, and NSPT are also specified by a "type" option and thus have as their defaults those of the default type (7204 for DC, 7242 for DP).

:SDEVICE This command specifies which peripheral devices are to be associated with a given symbiont. If used, this command must appear immediately prior to the :MONITOR command (see below) and immediately following the :STDLB (:CHAN/:DEVICE) commands. Only one :SDEVICE command may be used.

The :SDEVICE command has the form

```
:SDEVICE  (LMN, symbiont, name [,...])[, (...)]...
```

where

symbiont specifies the load module name of a system symbiont. Either ISSEG (input) or OSSEG (output) may be specified.

name specifies a peripheral device name of the form yyndd (see Tables 12, 13, and 14).

All peripheral devices must have been specified in :DEVICE commands (see above).

:MONITOR This command defines various Monitor and CPU parameters for the target system. It must be used and must appear immediately following the last command used to define the peripheral devices of the target system (i.e., either after the :SDEVICE command, if any, or else after the last :DEVICE or :STDLB command, whichever appears later in the deck).

The :MONITOR command has the form

```
:MONITOR (option) [, (...)]...
```

where the options are as follows. Note that the default values given are also the minimum that may be specified for each parameter.

SFIL, n specifies, in decimal, the number of job files that can be maintained by symbionts. One such file is needed for each batch job in the job queue and two files are needed for each job that has been executed but whose output has not yet been listed or punched. The default is 20.

TSTACK, size specifies, in decimal, the number of words in the Monitor's temp stack. The default is 200 for nonsymbiont systems and 250 for symbiont systems. The Monitor saves its environment as well as the TEMP area during interrupt processing. Thus, the more interrupts there are, the more TSTACK space is required. It is recommended that the user specify an additional 50 words for each external interrupt level and 50 words for BTM (if applicable).

CORE, size specifies, in decimal units of K (where k = 1024), the size of core storage in the target system. The default is 24.

QUEUE, size specifies, in decimal, the maximum number of I/O operations that may be queued at any one time. The default is 4. For a symbiont system, it is recommended that at least 20 be specified (to allow concurrent typing and execution).

MPOOL, size specifies, in decimal, the number of 34-word buffers to be pooled for use by the Monitor. The default is 3. For maximum I/O throughput, the number of MPOOL buffers should be equal to the maximum number of DCBs that may be open at any one time. For device I/O, one buffer is reserved for the OC device.

FQUEUE, value specifies, in decimal, the number of queue entries reserved for foreground programs. The value specified must be less than the number of entries defined for QUEUE (see above). The default is 0 if omitted (or incorrect).

FMPOOL, value specifies, in decimal, the number of Monitor buffers reserved for foreground use.

The value specified must be less than the number specified for MPOOL (see above). The default is 0 if omitted (or incorrect).

SPOOL, size specifies, in decimal, the number of 256-word buffers to be pooled for use by symbionts. The default is 0. SPOOL has no meaning for a nonsymbiont system. To obtain optimum efficiency, the number of SPOOL buffers must be equal to the number of CPOOL buffers (see below) plus 2 (one for the input symbiont and one for the output symbiont).

CPOOL, size specifies, in decimal, the number of 40-word buffers to be pooled for symbiont context block use. The default is 0. CPOOL has no meaning in a nonsymbiont system. To obtain optimum efficiency, there must be one context buffer per symbiont device type plus one per symbiont device.

There must be one SPOOL and one CPOOL per symbiont device type.

CFU, value specifies, in decimal, the number of 19-word buffers to be pooled for current file users. The default is 2. The minimum number of buffers specified for this pool should be the number of files (DCBs) that may be in use at any one time. Six are required if System Generations are to be performed.

ORG, value specifies, in hexadecimal, the load origin of the Monitor. This value should be the address of the first unused, even-numbered interrupt, since the Monitor must be biased above the highest interrupt. If there are no special interrupts, the origin should be at location 60_{16} which is the default value.

MPATCH, size specifies, in decimal, the number of word locations to be reserved for modification of the Monitor (i.e., a patch area). The default is 0.

:DLIMIT This command is used to specify the system parameters (default limits) that are to be associated with each job to be processed. All parameters omitted from or not specified explicitly by either a :LIMIT control command for a particular job or by a :DLIMIT command are set to their default limit. :DLIMIT must be used but can be null. It must follow the :MONITOR command.

Standard SYSGEN Default Limits

Job execution	15 minutes
Job processor listing output	100 pages
Object records	500 records
Job diagnostic output	100 pages
Executing programs output	100 pages

Standard SYSGEN Default Limits (cont.)

Temporary disk storage	256 granules
Permanent disk storage	256 granules
Index buffers	Two 256-word buffers
File buffers	Two 512-word buffers
Scratch tapes	Two tapes

The M:DLIMIT library load module has been modified to accommodate individual default limits for different priority settings (see PRTY option).

Only one :DLIMIT command is allowed. However, continuation is allowed on following cards when necessary.

The :DLIMIT command has the form

```
:DLIMIT [(option)][, (option)]...
```

where the options are

TIME, value specifies, in decimal, the default limit for job execution time. Value is expressed in minutes. If unspecified, the value 15 is assumed.

LO, value specifies, in decimal, the default limit for the number of pages to be listed by all processors involved in running a job. If unspecified, the value 100 is assumed.

PO, value specifies, in decimal, the default limit for the number of object records produced in running a job. If unspecified, the value 500 is assumed.

DO, value specifies, in decimal, the default limit for the number of pages of diagnostics produced in running a job. If unspecified, the value 100 is assumed.

UO, value specifies, in decimal, the default limit for the number of pages that may be output by the executing program(s) in a job. If unspecified, the value 100 is assumed.

TSTORE, value specifies, in decimal, the default limit for the number of granules (512 words) of temporary disk storage that may be used by a job. If unspecified, the value 256 is assumed.

PSTORE, value specifies, in decimal, the default limit for the number of granules of permanent disk storage that may be used by a job. If unspecified, the value 256 is assumed. The default limit is also constrained by the value in the user's authorization file.

IPOOL, value   specifies the (decimal) default number of 256-word buffers to be pooled for batch file indexing. If unspecified, the value 2 is assumed. Each open file requires an index buffer. If an insufficient number of index buffers exists, they will be shared (at the price of reduced system performance). This default can be overridden by use of a POOL control command at run time (see Chapter 2 of the BPM/Reference Manual, 90 09 54).

FPOOL, value   specifies the (decimal) default number of 512-word file blocking buffers to be allocated to batch tasks. If unspecified, the value 2 is assumed. In general, each open file requires a blocking buffer. If an insufficient number of blocking buffers exists, they will be shared (at the price of reduced system performance). This default can be overridden by use of a POOL control command at run time (see Chapter 2 of the BPM/Reference Manual 90 09 54).

SCRATCH, value   specifies, in decimal, the default limit for the number of scratch tapes that may be in use at any one time. If unspecified, the value 2 is assumed.

PRTY, value   specifies the priority (hexadecimal value) to be used for all following options to the next PRTY option or the end-of-record. All options preceding the first PRTY option will specify the values to be used for all limits not specified explicitly by priority. For example,

```
"DLIMIT  (LO,30), (PRTY,6), (LO,50),
         (PRTY,7), (PO,12)
```

All defaults except LO and PO are set to their standard SYSGEN default limits. LO limits will be set to 30 for all priorities except 6, which will be set to 50. PO limits will be set to their standard SYSGEN default limit (500 records) for all priorities except 7, which will be set to 12.

:ABS   This command must be used to specify which processors, if any, are to be entered into the system in absolute format and the size of an absolute file area for fast access to temporary disk storage. Processors entered in this manner will be managed as part of the system, thereby allowing a direct fetch of the processor. Only one :ABS command may be used.

The :ABS command has the form

:ABS[, size] [(proc$_1$[, S]) [, (proc$_2$[, S])] . . .]

where

proc$_i$ specifies the name of a processor to be assigned an absolute disk address.

S   specifies that the load module form of the processor is to be saved rather than deleted from the system. System Generation will always save the load module form of an overlaid processor that has been declared in the :ABS control command. The root of the tree structure is the only portion of the processor affected by the :ABS control card; therefore, for an overlaid processor, the load module must be saved. System disk storage space requirements are greatly reduced when nonoverlaid processors are released.

size   is the (decimal) number of words desired for the absolute storage area on disk. The default is 1024 (this is the minimum required to load the Monitor).

:FRGD   This command defines the foreground characteristics of the target system. Only one :FRGD command may be used.

The :FRGD command has the format·

:FRGD (option) [, (option)] . . .

where the options are:

NFRGD, value[†]   specifies, in decimal, the maximum number of foreground programs known to the Monitor at one time. This includes foreground programs which are absolute on disk, resident in memory, or both.

CT, address[†]   specifies the absolute hexadecimal interrupt location to which the Monitor's control task is to be connected. The control task is a resident Monitor routine that handles all unsolicited key-ins, background checkpoint and the processing of I/O cleanup and other real-time processes.

FCOM, size   specifies the number (decimal) of pages to be reserved for common reference by all foreground tasks. The default size is 0.

RESDF, size   specifies, in decimal, the number of pages to be reserved for resident foreground storage. The default is 0.

FIPOOL, value   specifies, in decimal, the number of 256-word buffers to be pooled for use in foreground file indexing. The default is 0.

FFPOOL, value   specifies, in decimal, the number of 512-word buffers to be assigned for use in file management (to be used by the Monitor in the packing and unpacking of foreground data). The default is 0.

NINT, value   specifies, in decimal, the maximum number of interrupts that will be used at one time.

[†] This parameter must be specified.

This includes the clock and external interrupts, but does not include the external interrupt for the Monitor's control task. The default is 0.

INTS, (name, size, pages)[, (. . .)]. . . specifies the names of foreground programs that are to be made absolute and loaded when the Monitor is booted from a PO or BI tape. These programs are also loaded when the Monitor is booted from disk. The "size" specifies the additional number (decimal) of words to be allocated on disk for expansion if the program is updated, "pages" specifies the additional (decimal) number of pages to be allocated for the program when it is loaded into memory. The load modules for these programs must have been formed at SYSGEN time by the relocating loader. Also, when they are loaded they are given control at their end-transfer address so that they can initialize themselves.

CTQ, value specifies, in decimal, the number of entries in the Monitor's control task queue. These queue entries are used for queuing up M:ABSLOAD and M:SBACK CALs. The default is 2.

:BTM This command specifies the parameters of the on-line time-sharing portion of the target system. Only one :BTM command may be used.

The :BTM command has the form

```
:BTM [(option)] [, (option)]. . .
```

where the options are shown in Table 17.

Table 17. :BTM Command Options

Option	Meaning	Default	Limits	Units
NUMUSERS, n	Total number of time-sharing consoles that may be in use at one time.	8	1-64	Dec.
USERSIZE, n	Size, in words, of the time-sharing memory area. The size must be a multiple of 512 (one page).	16384	12288- 65536	Dec.
NUMSYSTS, n	This provides an upper limit on the number of subsystems that can be in the system. It is prudent to provide for more than the standard set, so they can be added after the system is generated. This should also be taken into account when allocating swap area.	12	10-30	Dec.
BTMPM	The core allocation for tables used by the system tuning program.	---	---	---
BPMQTM, n	The minimum amount of time BPM runs before time-sharing users are scheduled.	200	10-500	ms.
BTMQTM, n	The maximum amount of time an on-line user may run before BPM receives another quantum.	200	50-500	ms.
BTMQTM2, n	The maximum amount of time a compute bound user may run before BPM receives another quantum.	800	50-2000	ms.
IBUFSIZE, n	COC input buffer size. The number of bytes that may be typed ahead before data is lost. This is also the maximum number of characters that may be typed before an activation character is typed.	100	80-200	Dec.
OBUFSIZE, n	The number of characters which can be held in each user's COC output buffer without further program intervention.	100	80-255	Dec.
IINT, n	The location of the COC input interrupt.	60	60-13F	Hex.
OINT, n	The location of the COC output interrupt.	61	61-13F	Hex.

:INTLB This command provides the capability of associating a label with an interrupt location. The label may then be used in the Monitor CALs such as M:ARM. More than one label and location pair may be specified.

The :INTLB command has the format

```
:INTLB (label, loc)[, (...)]...
```

where

label specifies a one- or two-character alphanumeric label.

loc specifies the absolute hexadecimal interrupt location to be associated with the label. The location must be greater than 5F and less than 140. Labels may also be associated with counters 1, 2, and 3.

If used, the :INTLB command must immediately follow the :FRGD command.

PASS2 CONTROL COMMAND SEQUENCE

The PASS2 control command sequence must be ordered according to the following scheme and must be first:

Case 1	Case 2	Case 3
[STDLB]	CHAN	CHAN
CHAN	[STDLB]	DEVICE
DEVICE	DEVICE	[STDLB]
[SDEVICE]	[SDEVICE]	[SDEVICE]
MONITOR	MONITOR	MONITOR
DLIMIT	DLIMIT	DLIMIT

where STDLB and SDEVICE are optional.

One other PASS2 control sequence which must be adhered to is:

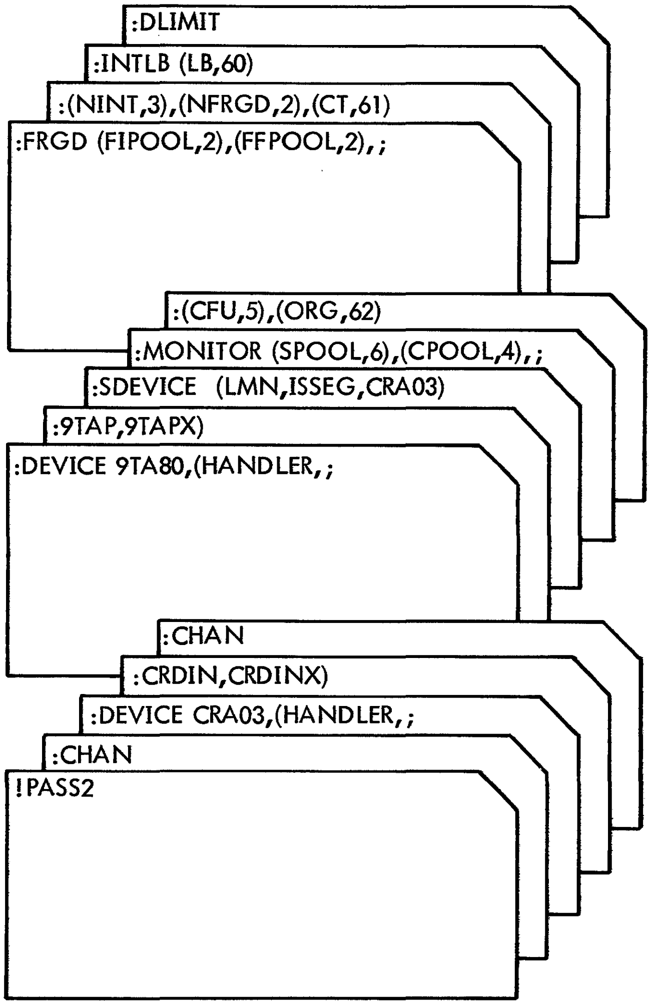
FRGD

[INTLB]

where INTLB is optional, but if it is needed, it must immediately follow the FRGD control command: this combination may appear anywhere following the MONITOR control command.

When a PASS2 control command parameter has a default, this implies that when a value is specified, the default given is the minimum accepted (unless otherwise specified) as well as being the default.

PASS2 EXAMPLE



```
:DLIMIT
:INTLB (LB,60)
:(NINT,3),(NFRGD,2),(CT,61)
:FRGD (FIPOOL,2),(FFPOOL,2),;
:(CFU,5),(ORG,62)
:MONITOR (SPOOL,6),(CPOOL,4),;
:SDEVICE  (LMN,ISSEG,CRA03)
:9TAP,9TAPX)
:DEVICE 9TA80,(HANDLER,;
:CHAN
:CRDIN,CRDINX)
:DEVICE CRA03,(HANDLER,;
:CHAN
!PASS2
```

PASS2 MESSAGES

All PASS2 messages are output on the LL device. When PASS2 attempts to continue (unless otherwise specified), it will search for a closing parenthesis ")" and continue processing from that point on. When an error message implies an error within a processor, this could also mean that there is not enough core to generate the current load module (LM). Table 18 lists PASS2 messages.

Table 18. PASS2 Messages

Message	Description
General Messages	
***CC IGNORED, PREVIOUS CC OF THIS TYPE ACCEPTED	The current control command type has already been encountered and processed. Only one set of a specific type of command is allowed in a run of PASS2. PASS2 continues to the next control command.
***CC'S NOT ENCOUNTERED, BUT POSSIBLY NEEDED	The PASS2 commands following this message were not encountered during this PASS2 run.
.....END OF PASS2.....	The end of PASS2 has been reached. PASS2 exits to the Monitor.
.....PASS2 CCI IN CONTROL.....	PASS2 has been entered.
***UNKNOWN OR MISPLACED CC	The current control command is unknown. PASS2 continues to next control command.
***UNKNOWN TYPE - xxx USED	The Monitor type option was not present or not legal. "xxx" is replaced with the default type (of the Monitor that is running).
CHAN/DEVICE/STDLB Messages	
$	When $ appears without additional messages, it indicates that there is a syntax error. PASS2 tries to continue.
***'ABSF'/'BCHK' PREVIOUSLY DEFINED	A device control command has defined ABSF and/or BCHK and they have also been defined previously. PASS2 continues to the next control command.
***CHAN TABLE FULL	The CHAN control command has overflowed the allocated core area. Up to 32 CHAN commands are allowed. PASS2 tries to continue.
***DCT TABLE FULL	The core area allocated for the DCT tables (peripheral device information tables) was not large enough. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.
***DEVICE ENTRY TABLE FULL	The DEVICE control commands have overflowed the allocated core area. Up to 96 devices may be defined. PASS2 tries to continue.
***DEVICE TYPE yy ILLEGAL	A DEVICE control command yyndd field contained a "NO" or "MT" as its yy. PASS2 tries to continue.
***DISC ENTRY TABLE FULL	The DEVICE control commands defining disk type (i.e., yyndd is of DCndd or DPndd type) have overflowed the allocated area. Up to 32 disks may be defined. PASS2 tries to continue.
***HANDLER CLIST FULL	The core area allocated was not large enough for the CLIST (peripheral command list area) tables. Up to 32 handler definitions are allowed. More are allowed if standard handler names are used. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.

Table 18. PASS2 Messages (cont.)

Message	Description
***HGP CANNOT BE FORMED FOR yyndd	A DEVICE yyndd command (where yyndd is for a DC or DP device) contained a syntax error for which no defaults can be taken. PASS2 tries to continue.
***HGP TABLE FULL	The core area allocated for HGP tables for RAD or disk pack devices was not large enough. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.
***INSUFFICIENT PAGES AVAILABLE	The available core was not large enough for allocation required by generation of the load module. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.
***LOAD MODULE CANNOT BE GENERATED	The PASS2 processor for CHAN/DEVICE/STDLB contained an error. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.
***'NAME' OR SYNTAX INVALID	A CHAN control command option field had a syntax error or invalid name for the optimizer option or the DEVICE control command contained a syntax error or invalid name for the handler option. PASS2 tries to continue.
***NO CHAN/DEVICE INFO	No CHAN and DEVICE control commands have been encountered, although STDLB control commands have been processed. PASS2 tries to continue to the next command that is not a CHAN, DEVICE, or STDLB command.
***NO DEVICE FOR CHAN	A CHAN control command has been encountered without any device definitions for the channel. PASS2 tries to continue.
***NO DEVICE FOR TYPMNE yy (OPLBL=ll)	A STDLB command or a default has defined an operational label (ll) whose assignment is to type mnemonic yy for which there is no device definition. PASS2 tries to continue.
***NO DISC DEFINED	No RAD or disk pack was defined by a DEVICE control command. PASS2 continues to the next control command that is not a CHAN, DEVICE, or STDLB command.
***NO HANDLER NAME GIVEN	The handler option was not present when a device whose type mnemonic was unknown to PASS2 was defined. PASS2 continues to the next control command.
***ONLY PFA VALID ON PRIVATE DEVICES	A nonzero PER, ABSF, PSA, or BCHK option appeared with PRIVATE and has been ignored.
***OPLB xx EQUIVALENT yy MISSING	A STDLB control command specified that an operational label (xx) standard assignment was to another operational label (yy) that was not defined (although it may be defined later). PASS2 tries to continue.
***PER MUST BE GRANULE ALLOCATED - PER IGNORED	The PER option was used with either CYLINDER or PRIVATE.
***STDLB ENTRY TABLE FULL	The STDLB control command information has overflowed the allocated core area. Up to 128 standard label

Table 18. PASS2 Messages (cont.)

Message	Description
***STDLB ENTRY TABLE FULL (cont.)	definitions or up to 32 unique operational labels are allowed. PASS2 tries to continue.
***SUM OF PSA+PFA+PER+BCHK+ABSF>SIZE	There was a conflict in the summation of the given list of variables and the defined RAD or disk pack size. The message may appear several times for a given disk (i.e., if the conflict is determined after the summation of PSA+PFA, then the message will appear for this summation and once for each of the remaining summations). The message will not necessarily appear for BCHK or ABSF, although they may in themselves overflow the disk size. The processor continues.
***SYNTAX ERROR	This message appears in conjunction with the $ message. PASS2 tries to continue.
***THIS DISC ALREADY DEFINED	A DEVICE control command defined a disk device (i.e., yyndd) that was already defined. PASS2 tries to continue.
***TYPMNE ENTRY TABLE FULL	More than 64 type mnemonics have been specified by DEVICE control commands. These mnemonics are specified by yyndd specifications, where yy (e.g., TY) is other than the standard set of type mnemonics set forth in Table 14. PASS2 tries to continue.
***UNKNOWN DEVICE yyndd	The yyndd field of a DEVICE control command was invalid (i.e., bad syntax) or, for the STDLB command, the yyndd referenced was not defined by a DEVICE control command. PASS2 tries to continue.
***UNKNOWN DEVICE yyndd FOR ll	The yyndd field referenced by a STDLB control command has not been defined by a DEVICE control command. ll is the operational label. PASS2 tries to continue.
***VALID 'CHAN' CC MUST PRECEDE 'DEVICE' CC	A DEVICE control command not preceded by a CHAN control command was encountered. PASS2 tries to continue.
SDEVICE Messages	
INVALID 'yyndd'	The yyndd field was either unknown (i.e., not defined by a device control command) or the syntax of yyndd was in error. PASS2 tries to continue.
INVALID KEYWORD	The keyword was not LMN. PASS2 tries to continue.
INVALID SYMBIONT NAME	The symbiont name was not alphanumeric or was greater than seven characters long. PASS2 tries to continue.
MODIFY ERROR	There was an error in the processor. PASS2 exits to the Monitor.
NO ROOM LEFT FOR :SDEVICE	The generated load module was too large for allocated core area. PASS2 exits to the Monitor.
REMAINDER OF CC IGNORED	A syntax error occurred and PASS2 cannot recover by finding a closing parenthesis ")". PASS2 finishes generating the load module.

Table 18. PASS2 Messages (cont.)

Message	Description
'SDEVICE ABORTED'	This message appears in conjunction with other messages for conditions that cause an exit to the Monitor.
SYNTAX ERROR	The SDEVICE control command contained a syntax error. PASS2 tries to continue.
MONITOR DLIMIT Messages	
***ERROR IN PROCESSOR – JOB ABORTED	The processor is in need of repair. PASS2 error exits to the Monitor.
***ILLEGAL TYPE OF SIZE.	The value for a PRTY option for DLIMIT was not a positive hexadecimal value, exceeded the current maximum (F), or was previously specified. All options between the offending PRTY option and the next PRTY (or end-of-record) are processed for syntax errors and are ignored when the M:DLIMIT library module is generated.
***ILLEGAL TYPE OR SIZE	A value was the wrong type (decimal, hexadecimal, or ndd) or was too large or too small, or a text string contained too many characters. PASS2 ignores the value and continues with the next character unless the next character should be a closing parenthesis, in which case PASS2 searches for the next opening parenthesis.
***INADEQUATE CORE SPACE – SKIP TO NEXT CC	The load module cannot be generated in the available core space.
***INVALID, UNKNOWN, OR DUPLICATE KEYWORD	A keyword was invalid, unknown, or duplicate. PASS2 searches for the next opening parenthesis '('.
***MODIFY ERROR – SKIP TO NEXT CC	The load module cannot be generated in the available core space.
***SYNTAX ERROR – 'x' EXPECTED	PASS2 expected (x) a closing parenthesis, a comma, or an opening parenthesis, but did not encounter it. PASS2 searches for the next opening parenthesis.
Monitor Messages	
**FMPOOL>=MPOOL, FMPOOL IGNORED	The FMPOOL option value is equal to or greater than that of the MPOOL option. PASS2 ignores the field and attempts to continue.
**FQUEUE>=QUEUE, FQUEUE IGNORED	The FQUEUE option value is equal to or greater than that of the QUEUE option. PASS2 ignores the field and attempts to continue.
ABS Messages	
'(' EXPECTED BUT NOT FOUND	There are no ABS processor definitions. This is not catastrophic. PASS2 continues.
')' EXPECTED BUT NOT FOUND	A closing parenthesis was expected but not found. PASS2 tries to continue.
'ABS' ABORTED	This message appears with messages that result in an exit to the Monitor.

Table 18. PASS2 Messages (cont.)

Message	Description
INVALID PROCESSOR NAME	The processor name is not alphanumeric. PASS2 tries to continue.
INVALID SIZE OR SIZE MISSING, DEFAULT TAKEN	The ABS control command normally defines a size, but the field may be left blank if the default (1024) is desired. Also, if a size is defined and it is valid, e.g., <1024, then the default will be taken. PASS2 continues.
':L' NAME ILLEGAL OR NAME ALREADY DEFINED	A name of a processor cannot be ':L' and may not be defined more than once. PASS2 tries to continue.
LOAD MODULE GEN. UNSUCCESSFUL	There was an error in the ABS processor. PASS2 exits to the Monitor after it displays the abort message.
NO FIELDS ON CC	Only the ABS control command with no parameters was included. This is not catastrophic but is the minimum required to satisfy the mandatory ABS control command. PASS2 continues.
NO PAGES AVAILABLE	Not enough core was available for generation of load modules. PASS2 exits to the Monitor.
PROCESSOR NAME>11 CHARACTERS	A processor name was greater than 11 characters in length. PASS2 tries to continue.
'S' EXPECTED BUT NOT FOUND ** 'S' ASSUMED	The optional 'S' field contained some character(s) other than S. PASS2 continues.
SYNTAX ERROR	A terminator was encountered but not recognized as a legitimate terminator for syntax purposes.
FRGD/INTLB Messages	
***CT FIELD NOT=>60 OR=13F, PROC. ABORTED	The CT option was invalid or missing. PASS2 exits to the Monitor.
***DELIMITER ERROR	An unknown delimiter was encountered in an option field. PASS2 tries to continue.
***DELIMITER ERROR, PROCESSOR ABORTED	An unknown delimiter appeared between option fields. PASS2 exits to the Monitor.
***GEN. OF LM UNSUCCESSFUL	The FRGD/INTLB processor contained an error. PASS2 exits to the Monitor.
***INVALID DECIMAL VALUE	An option field did not contain a valid decimal number. PASS2 tries to continue.
***INVALID HEXADECIMAL VALUE	An option field did not contain a valid hexadecimal number. PASS2 tries to continue.
***NAME INVALID OR > 11 CHAR. OR > 2 CHAR.	A name was either nonalphanumeric or was too long. PASS2 tries to continue.
***NFRGD FIELD MISSING OR INVALID, PROC. ABORTED	The NFRGD option was invalid or missing. PASS2 exits to the Monitor.

Table 18. PASS2 Messages (cont.)

Message	Description
***NOT ENOUGH CORE AVAILABLE TO GEN LM, PROC. ABORTED	There was not enough core available to generate the load module. PASS2 exits to the Monitor.
***SIZE/PAGES VALUE INVALID	The value defined was in error. PASS2 tries to continue.
***UNKNOWN KEYWORD	An option field contained an unknown keyword. PASS2 tries to continue.
***VALUE ERROR, DEFAULT TAKEN	An option field violated the default. PASS2 tries to continue.
BTM Messages	
':BTM' ABORTED	PASS2 has made an abort exit to the Monitor.
EXPECTED ALPHANUMERIC STRING NOT FOUND	An alphanumeric string is invalid (e.g., nonalphanumeric). PASS2 tries to continue.
EXPECTED DELIMITER NOT FOUND	An improper delimiter was encountered. PASS2 tries to continue.
EXPECTED NUMERIC FIELD NOT FOUND	A numeric field was invalid, e.g., nonnumeric. PASS2 tries to continue.
NO MORE SPACE LEFT FOR ':BTM'	A load module cannot be generated in the available core space. PASS2 makes an abort exit to the Monitor.
PARAMETER UNKNOWN OR TOO LONG	A parameter field is unknown or a field is too long. PASS2 tries to continue.
VALUE OUT OF LEGAL BOUNDS	A value was found that is either too small or too large. PASS2 tries to continue.

LOCCT PROCESSOR

The LOCCT processor provides an optional phase of system generation that generates a file defining the elements and load structure of a user's processor or Monitor. LOCCT may be called during system generation to create, in the current account, a permanent RAD or disk pack file containing the LOCCT, ROM, and TREE tables for a given set of LOAD (!LOCCT) and TREE control commands defining the load structure of a user's processor or Monitor. It also outputs a copy of this file to the PO device. The contents of the LOCCT-generated file are referred to as "LOCCT tables". Each record consists of a binary card image having the format shown in Figure 34.

The LOCCT processor is entered via the following control command sequence:

```
!LOCCT(LMN,...)...(EF,(...))†
!TREE ...
!DATA
:LOCCT name
```

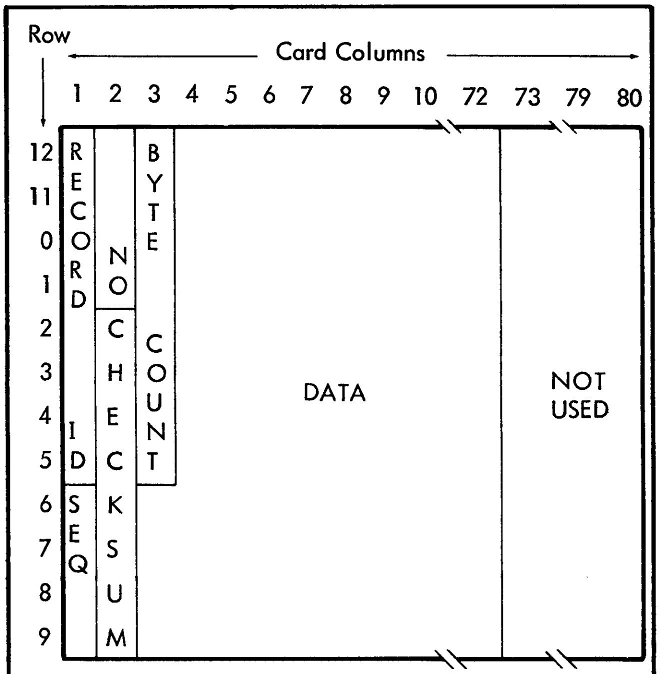
where name specifies the name desired for use in retrieving the LOCCT file from RAD. This name must be no longer than ten characters.

Continuation cards are not allowed for the :LOCCT command. If comments are desired, they must be preceded by a period.

The "LOCCT name" command (data card) must immediately follow the !DATA command so that the Control Command Interpreter (CCI) will know that the LOCCT process is to be entered.

If the PASS3 processor is to be used to load a standard program at some later time, the LOCCT processor must be used once for every unique set of LOAD (!LOCCT command) and TREE control commands defining the load structure of a

[†] This control command replaces the !LOAD command for this type of process and contains the same information the !LOAD command would normally contain.



where

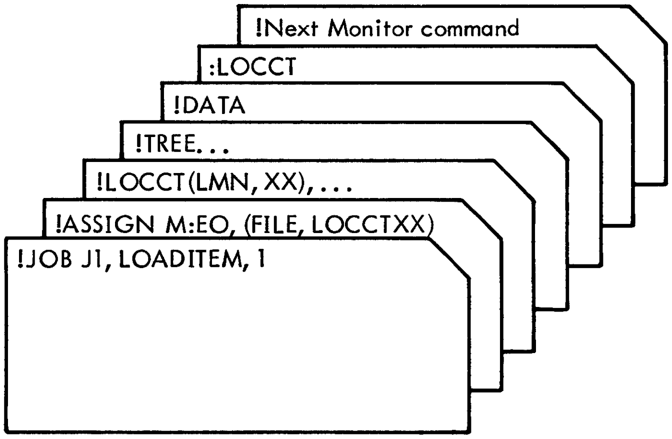
RECORD ID	X'3E' binary card code and X'1E' binary end card code
SEQ NO	two-digit (hex) sequence number
CHECKSUM	byte checksum of card image
BYTE COUNT	Number of useful bytes in card image, including control word in columns 1-3

Figure 34. LOCCT Record Format

processor or Monitor. It should be executed in the account in which PASS3 will eventually be executed. Also, all element file names in the LOAD (!LOCCT) command should originate in the account in which PASS3 will be executed. The file name used to generate the LOCCT file is determined by appending ~~the "name" from the control command~~ (see above) to the characters "LOCCT" (for the processor METASYM, the LOCCT file name would be LOCCTMETASYM). The name is optional. However, if the name does not appear, the M:EO DCB must have been previously assigned to the file name that PASS3 will reference later for a particular LOCCT. If a name appears, and an ASSIGN command assigns M:EO to a file name, the name on the LOCCT control command takes precedence. When the file name is determined via an ASSIGN command, the file name must include the first five characters (LOCCT) as part of the file name.

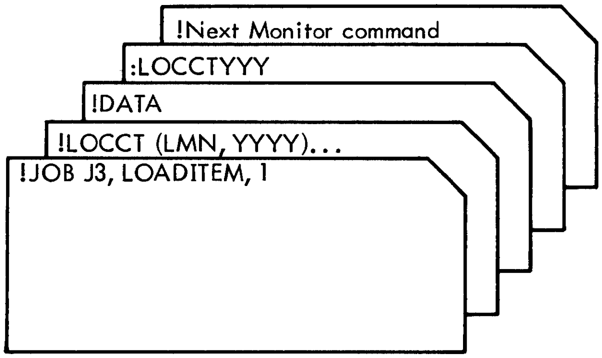
LOCCT EXAMPLES

Examples for using the LOCCT processor are shown below.



```
!JOB J1, LOADITEM, 1
!ASSIGN M:EO, (FILE, LOCCTXX)
!LOCCT(LMN, XX), . . .
!TREE. . .
!DATA
:LOCCT
!Next Monitor command
```

This example will generate a permanent file, LOCCTXX, that will contain the LOCCT, ROM, and TREE tables for the first job's LOAD (!LOCCT) and TREE commands. The file will be under the account number "J1". A permanent copy will also be output to the PO device. This example's file name is determined by the "ASSIGN M:EO" control command.



```
!JOB J3, LOADITEM, 1
!LOCCT (LMN, YYYY). . .
!DATA
:LOCCTYYY
!Next Monitor command
```

This example will generate a permanent file, LOCCTYYYY, with the information from the LOAD (!LOCCT) command. The file will be under the account number "J3". A permanent copy will also be output to the PO device.

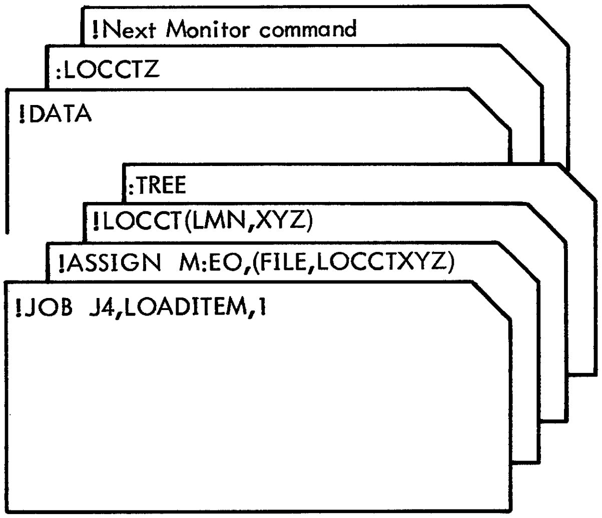
The example below will generate a permanent file, LOCCTZ using the information from the LOAD (!LOCCT command) and TREE commands. The file will be under the account number "J4". A permanent copy will also be output to the PO device. Notice that the ASSIGN command's file name is ignored and also that the LOCCT file name need not be the same as the load module name.

LOCCT MESSAGES

Table 19 lists the messages that are output during LOCCT processing. All messages are output on the LL device.

Table 19. LOCCT Messages

Message	Description
***ABS READ ERR/ABN=xxxx	An I/O error or abnormal condition has been encountered on the ABS read request. The value xxxx is the I/O error code.
***CANNOT GENERATE LOCCT WITH ROMS ON LABELED TAPE	An element file is on labeled tape.
***I/O ERR/ABN FOR READ C=xxxx	An I/O error or abnormal condition has been encountered on the C device. The value xxxx is the I/O error code.
***I/O ERR/ABN FOR WRITE EO=xxxx	An I/O error or abnormal condition has been encountered on the EO device. The value xxxx is the I/O error code.
***I/O ERR/ABN FOR WRITE PO=xxxx	An I/O error or abnormal condition has been encountered on the PO device. The value xxxx is the I/O error code.
LOCCT PROCESSOR ABORTED	This message is output after other LOCCT messages. LOCCT then exits to the Monitor.
***NAME INVALID	The name in the LOCCT command was in error.
***NAME>10 CHARACTERS	The name in the LOCCT command was greater than ten characters in length.
***OPEN EO ERR/ABN=xxxx	An I/O error or abnormal condition has been encountered by LOCCT while trying to open the EO device. The value xxxx is the I/O error code.
***ROM TABLE END CANNOT BE FOUND	The ROM table is invalid.
***UNKNOWN CC OR CONTINUATION ILLEGAL	The name of the LOCCT command entered was invalid or the LOCCT command was to be continued. LOCCT displays the abort message and then exits to the Monitor.



```
!Next Monitor command
:LOCCTZ
!DATA
:TREE
!LOCCT(LMN,XYZ)
!ASSIGN  M:EO,(FILE,LOCCTXYZ)
!JOB  J4,LOADITEM,1
```

The permanent file name may be any name desired, as long as LOCCT and PASS3 both reference the same name for a given LOCCT file.

PASS3 PROCESSOR

The purpose of this processor is to cause the loading of standard Monitors, processors, and libraries automatically via preestablished LOAD (OVERLAY or OLAY) and TREE structures. These structures must have been generated previously by the LOCCT processor. PASS3 is entered via the control command.

```
!PASS3 [type[, option]]
```

where

type	is the system type being generated (BPM, BTM, UTM). The default is the system under which PASS3 is running.

MON	causes an abort if M:MON cannot be loaded successfully or if it has not been loaded when attempting to load something else.

ALL causes an abort if anything cannot be loaded successfully, either because the loader finds errors or because M:MON does not exist.

If no option is specified, the error conditions will be ignored.

The commands that control PASS3 have the form

```
:id [(option[,option])]
```

where

id is the name of a LOCCT information table that is to be obtained to define the load structure of a Monitor, processor, or library subroutine (e.g., X, 9EDIT, FMGE). The id must not be longer than 10 characters.

option is optional information used to modify the default LOAD (OVERLAY or OLAY) command structure in the LOCCT table, or may consist of general information to PASS3 (see the list of parameters below). The option field may be continued on continuation cards if necessary.

A PASS3 control command identifies a standard system (Monitor), processor, or library subroutine name for which a LOCCT table is to be obtained. This LOCCT table will describe to the loader how the named routine is to be loaded. The LOCCT will be assumed to be in the current account number, unless a previous ASSIGN command assigned the M:EI DCB to some other account number. PASS3 will accept LOCCT tables from only one account.

The control commands may contain optional parameters. The possible parameters are as follows:

BIAS = value specifies that the load bias in the LOCCT table for the routine named "id" is to be changed to the specified hexadecimal "value". The bias is converted to the next higher page boundary, if not already at a page boundary. The maximum bias value may not exceed X'1FFFF'. If a bias offset is specified (see "BIAS = + offset" below), the "offset" will be added to the bias "value".

BIAS = + offset specifies that a hexadecimal "offset" is to be added to the specified bias "value" (see "BIAS = value" above). If no BIAS = value has been specified, the offset will be added to the lower limit of the background area (BKGRDLL) if the M:MON load module is present in the current account. If M:MON is not present and no BIAS = value has been specified, the offset value has no effect and the LOCCT bias is unchanged. The logical interation of these bias values is illustrated below.

BIAS = value	BIAS = +offset	M:MON	Resulting Bias
unspecified	unspecified	absent	LOCCT unchanged
unspecified	unspecified	present	BKGDRLL
unspecified	specified	absent	LOCCT unchanged
unspecified	specified	present	BKGRDLL + offset
specified	unspecified	absent	BIAS = value
specified	unspecified	present	BIAS = value
specified	specified	absent	BIAS = value + offset
specified	specified	present	BIAS = value + offset

SAVE $(name_1, name_2, \ldots, name_n)$ specifies that the named element files will not be deleted. All options not SAVEd will be deleted (see "DELETE", below).

DELETE specifies that when the loader has completed the loading of the standard system defined by "id", all element files comprising this module will be deleted from the RAD or disk pack, except for those specifically SAVEd (see "SAVE" above).

The deleted files must be in the current account, must not be protected by a password, and must be on disk rather than labeled tape. Unless SAVEd, the LOCCT table will also be deleted.

When the LOCCT table for a particular id[†] has been obtained and modified, PASS3 will write it to the ABS scratch area on disk in the executing Monitor system. This area is the communication area between PASS3 and the loader. PASS3 will then do a M:LINK call on the Monitor requesting the loader (LOADER) as its overlay. The loader must appear as a load module file in the :SYS account. When the loader completes its function, it will do a M:LDTRC call on the Monitor requesting a return to the calling program; namely, PASS3. At this time, the LOCCT table and all element files comprising the load module will be deleted if the DELETE and/or SAVE were encountered in the control command and if the load was successful. PASS3 will then continue to its next control command.

PASS3 will also set up automatic biasing once the load module M:MON (Monitor) has been loaded under the current account number. This is accomplished by obtaining the tree structure for M:MON, and searching for the end of the longest overlay path. Once this is obtained, it will be

[†] The specified id is used to form the file name of a LOCCT table to be obtained. The id will be appended to the characters "LOCCT". This implies that for an id of FMGE, the LOCCT file name will be LOCCTFMGE.

used to set the bias in each LOCCT prior to entering the Loader, unless a BIAS is specified on a PASS3 control command at which time the specified bias is used to modify the referenced LOCCT only. If a bias is desired which is less than the bias determined from the M:MON load module then a warning message will be displayed. If neither bias is found, the LOCCT table will retain the bias specified on the original !LOAD control command, and this bias will be displayed. PASS3 will also display the automatic bias which is determined from the M:MON load module.

All Monitors should be loaded by PASS3 to take advantage of PASS3's ability to form the HANDLERS file automatically. If a Monitor is to be loaded and there is no LOCCT for it, the LOAD (!LOCCT)/TREE commands must be used to generate a LOCCT by the LOCCT processor, and then PASS3 can use the LOCCT tables for the loading of the Monitor.

PASS3 will obtain the information in SPEC:HAND, the file that SYSGEN PASS2 generated. The data in this file will identify which I/O handlers are required for this Monitor (M:MON load module). If the SPEC:HAND file does not exist, PASS3 will abort the loading of the M:MON load module only, and will continue to the next LOCCT.

After obtaining the data from the SPEC:HAND file, PASS3 will open BASHANDL (the basic handler's file) as an input file and the file HANDLERS as an output file. The BASHANDL file will be copied to the new file HANDLERS and the BASHANDL file will be closed and saved. PASS3 will then obtain a handler name from the SPEC:HAND file, will open that file for input, and will copy the file to the new HANDLERS file. If a handler name from the SPEC:HAND file is found to be a part of the basic handlers file (BASHANDL), it will be ignored (see Table 20). When all of the handlers required have been merged into the HANDLERS file, this file will be closed and saved. All files accessed by PASS3 by this technique will be saved. PASS3 will then proceed to link to the overlay loader. If any handler cannot be found, PASS3 will abort the loading of the M:MON load module only, and will continue to the next LOCCT.

Table 20. Handlers in BASHANDL File

Name	Device
KBTIO	TY
CRDIN	CR
PRTOUT	LP
PRTOUTL	LP (low cost)
DISCIO	DC
MTAP	9T, MT
7TAP	7T

PASS3 TREE STRUCTURE ANALYZER

When PASS3 encounters a LOCCT table for a Monitor, i.e., the load module name in the LOCCT is "M:MON", PASS3 will proceed by analyzing the tree structure. A load module named "ROOT" will be generated and will contain the Monitor's variable overlay structure tables and the defined segment numbers for each Monitor overlay according to the given tree (overlay) structure. The load module is generated prior to requesting the loader. The segment number definitions are generated by taking each name in the LOCCT tables tree structure, appending the characters "SEG" to this name, and assigning it a number (e.g., for the name EXIT, the segment name will be EXITSEG). Therefore, a new overlay segment can be added to the Monitor's tree structure and will be referenced by its name with the appended characters "SEG" (e.g., the new segment AX is added to the Monitor tree structure and will be referenced by the segment name AXSEG). The LOCCT tables tree contains only one name for each overlay defined on the original tree control command. This name is the first name only of a series. That is, if the tree structure is

```
!TREE A-B-(C, D-E)
```

the names in the LOCCT tables tree would be

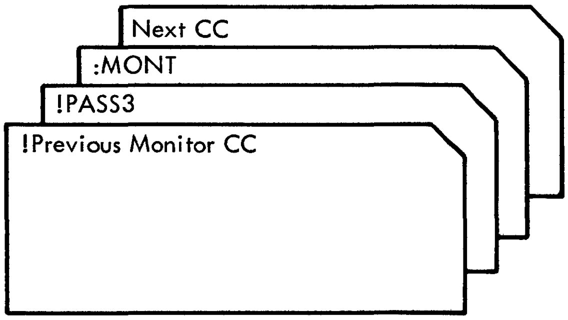
```
A, C, D
```

and the segment names would be

```
ASEG, CSEG, DSEG
```

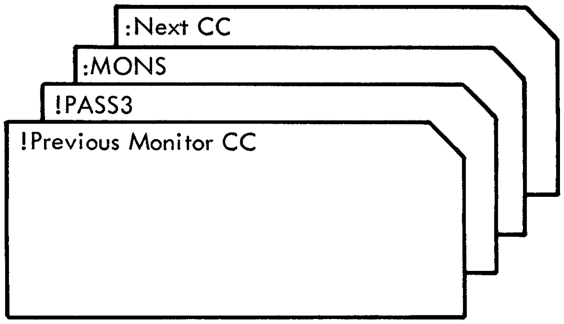
Note: All Monitors must be loaded by PASS3. A Monitor which is loaded by !LOAD/!TREE control commands will not contain all of the information for it to execute. If a Monitor is to be loaded and there is no LOCCT for it, the LOAD/TREE commands must be used to generate a LOCCT by the LOCCT processor, and then PASS3 can use the LOCCT tables for the loading of the Monitor. A Monitor tree structure may not contain more than five levels, i.e., four overlay levels plus the root.

PASS3 EXAMPLES



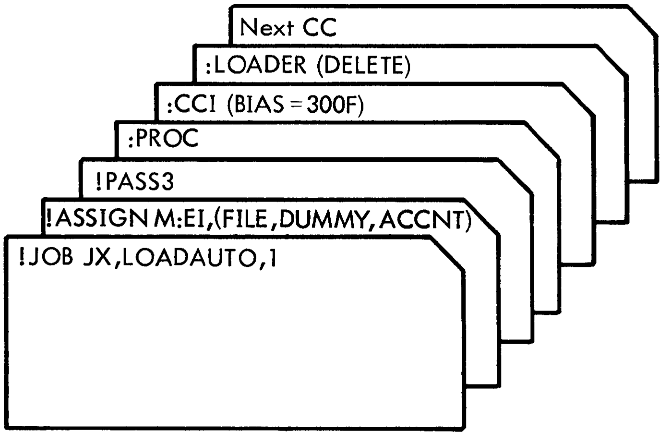
```
Next CC
:MONT
!PASS3
!Previous Monitor CC
```

This example will obtain the LOCCT table in the file LOCCTMONT. No bias change is specified and all element files comprising MONT will be saved. This load function might represent a minimal Monitor (M:MON). The LOCCTMONT file will be assumed to be in the account number under which this job is being run.



```
:Next CC
:MONS
!PASS3
!Previous Monitor CC
```

This example is similar to the previous one, the file name being LOCCTMONS. This load function might represent a symbiont Monitor system with load module name M:MON.

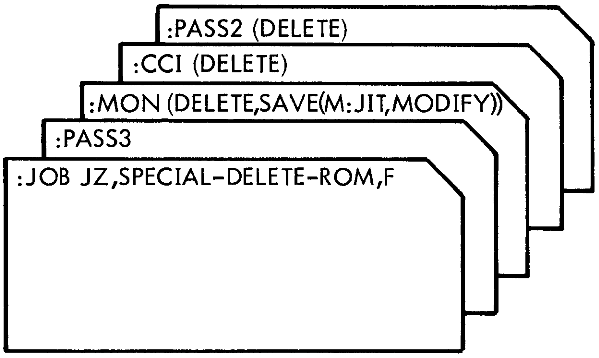


```
Next CC
:LOADER (DELETE)
:CCI (BIAS=300F)
:PROC
!PASS3
!ASSIGN M:EI,(FILE,DUMMY,ACCNT)
!JOB JX,LOADAUTO,1
```

This example will obtain the LOCCT tables for LOCCTPROC, LOCCTCCI, and LOCCTLOADER from the ACCNT account and not the job's account, JX. The new bias for CCI will become 3200, and all element files making up the LOADER will be deleted.

The recommended procedure when using PASS3 is to load the Monitor load module (M:MON) first. This will allow for truly automatic biasing of all future processors.

When PASS3 is being used to delete element files (DELETE option), certain restrictions must be adhered to. The Monitor (M:MON) and CCI have a common element (namely, M:JIT) and the Monitor and SYSGEN PASS2 have a common element (namely, MODIFY). The method for deleting the ROMs may be as follows:



```
:PASS2 (DELETE)
:CCI (DELETE)
:MON (DELETE,SAVE(M:JIT,MODIFY))
:PASS3
:JOB JZ,SPECIAL-DELETE-ROM,F
```

This procedure will load the Monitor (M:MON) with DELETE (except for M:JIT, and MODIFY), CCI with DELETE, and PASS2 with DELETE. There are other combinations similar to this which incorporate other processors.

Table 21 lists the messages that are output during LOCCT processing. All messages are output on the LL device.

Table 21. PASS3 Messages

Message	Description
**BIAS NOT HEXADECIMAL VALUE OR TOO LARGE VALUE	The bias value was invalid. PASS3 continues to the next control command.
****BIAS USED WILL BExxxxx	BKGRDLL is unknown and no bias was specified. The value xxxxx is the bias obtained from the LOCCT table and is for information only. PASS3 continues.
**BIN. CARD SEQUENCE ERR, SEQ. #xxxx	The LOCCT table contained a sequence number error. The value xxxx is the card sequence number where the error occurred. PASS3 continues to the next control command.

Table 21. PASS3 Messages (cont.)

Message	Description
**BIN. CARD INVALID TYPE, SEQ. #xxxx	The LOCCT table contained a sequence type error. The value xxxx is the card sequence number where the error occurred. PASS3 continues to the next control command.
**CANNOT OPEN/RELEASE	A SAVE or DELETE option was requested and a delete element could not be found. The file name that could not be found is output following this message. The message is output n-1 times, where n is the number of times an element file appears in a tree structure. Thus, the message does not indicate an error in many cases since the load module is built correctly and the element file is deleted the first time it is encountered. PASS3 continues to the next delete item.
**CC ERROR, NO ':' IN COLUMN 1	A PASS3 control command did not contain a ':' in column 1. PASS3 continues to the next control command.
**CC ID INVALID	The PASS3 control command's id is invalid. PASS3 continues to the next control command.
**CHECKSUM ERROR, SEQ. #xxxx	The LOCCT table contained a checksum error. The value xxxx is the card sequence number where the error occurred. PASS3 continues to the next control command.
**DELIMITER NOT (), = OR SYNTAX BAD	The syntax of a PASS3 command was either invalid or the expected delimiter should have been (), or =. PASS3 continues to the next control command.
**ID SIZE > 10 OR = 0 CHARACTERS	The id did not exist or was too large. PASS3 continues to the next control command.
**I/O ERR/ABN ON M:C = xxxx	An I/O error or abnormal condition was encountered while performing I/O on the C device. The value xxxx is the I/O error/abnormal code. PASS3 exits to the Monitor.
**KEYWORD NOT BIAS/DELETE/SAVE	A BIAS, DELETE, or SAVE keyword was expected but not found. PASS3 continues to the next control command.
**KEYWORD SAVE ALREADY USED	A second SAVE option was specified. PASS3 continues to the next contol command.
****M:MON BKGRDLL is xxxx	PASS3 is finished. The BKGRDLL for the Monitor loaded within this system generation is xxxx.
M:MON NOT SUCCESSFULLY LOADED	The MON or ALL option has been specified in a !PASS3 command and M:MON either cannot be loaded or has not been loaded. PASS3 aborts.
****M:MON TREE STRUCTURE >5 LEVELS	The M:MON tree structure contained more than five levels, including the root.
MODULE NOT SUCCESSFULLY LOADED	The ALL option was specified in a !PASS3 command and the loader found errors loading a processor. PASS3 aborts.
**NAME INVALID	A name defined by the SAVE option was invalid. The name must be alphanumeric. PASS3 continues to the next name.

Table 21. PASS3 Messages (cont.)

Message	Description
**OPEN M:EI ERR/ABN = xxxx (LOCCT)	An I/O error or abnormal condition was encountered during an open operation on the EI device. The value xxxx is the I/O error or abnormal code. PASS3 then exits to the Monitor.
**OPEN M:MON ERR/ABN = xxxx	An I/O error or abnormal condition was encountered while PASS3 was trying to open the M:MON load module. The value xxxx is the I/O error or abnormal code. PASS3 exits to the Monitor.
****OPEN/READ BASHANDLFILE ERR/ABN = xxxx	PASS3 could not obtain a given file when forming the HANDLERS file. The value xxxx is the error or abnormal code. BASHANDL names are replaced by the current file name being merged into the HANDLERS file. PASS3 continues to the next command.
****OPEN/READ SPEC:HAND FILE ERR/ABN = xxxx	PASS3 could not obtain a given file when forming the HANDLERS file. The value xxxx is the error or abnormal code. SPEC:HAND names are replaced by the current file name being merged into the HANDLERS file. PASS3 continues to the next command.
####PASS3--COMPLETED####	PASS3 returned to the Monitor.
####PASS3--IN--CONTROL####	PASS3 has been entered.
**READ M:EI ERR/ABN = xxxx (LOCCT)	An I/O error or abnormal condition was encountered during a read operation on the EI device. The value xxxx is the I/O error or abnormal code. PASS3 exits to the Monitor.
**READ M:MON ERR/ABN = xxxx	An I/O error or abnormal condition was encountered while PASS3 was trying to read the M:MON load module. The value xxxx is the I/O error or abnormal code. PASS3 exits to the Monitor.
**SPECIFIED BIAS < BKGRDLL	The specified bias is less than the background lower limit bias obtained from the M:MON load module. This is only a warning. PASS3 continues.
****UNKNOWN TYPE – xxxx USED	The Monitor type option was not present or not legal. "xxxx" is replaced with the default type (of the Monitor that is running).
**WRITE ABS ERR/ABN = xxxx (LOCCT)	An I/O error or abnormal condition was encountered while PASS3 was writing on the ABS scratch area on the RAD or disk pack. The value xxxx is the I/O error or abnormal code. PASS3 exits to the Monitor.

DEF PROCESSOR

The DEF processor is called upon to generate PO tapes containing a bootable Monitor system and all keyed files in the current account, and BO tapes containing the current Monitor system from the :SYS account and all consecutive files in the current account.

!DEF This command causes the Monitor to fetch the DEF processor from the RAD and to transfer control to it. The command has the form

```
!DEF[type][,version#]
```

where

type is the Monitor type:

BPM specifies that the bootable Monitor portion of the tape will be BPM/BTM.

UTS specifies that the bootable Monitor portion of the tape will be UTS.

version$^{\#}$ specifies a 3-character field defining the version number of the target system. Only the first three characters specified are used.

The following commands are DEF control commands.

:INCLUDE This command allows the user to write to tape files that have a different organization than those automatically written (CONSEC files to a PO tape, KEYED files to a BO tape). The command has the form

```
:INCLUDE (item,item,...)
```

where item identifies a file to be included on the tape. Table 22 contains a list of files automatically INCLUDEd.

Multiple :INCLUDE commands and continued commands are allowed.

All INCLUDEd files are put in the :SYS account when the system is booted.

:IGNORE This command causes DEF to avoid writing to tape a file that would otherwise be written because of its organization. The command has the form

```
:IGNORE (item,item,...)
```

where item is a file to be ignored. (LASTLM and SPEC:HAND are automatically IGNOREd from PO tapes.)

The :IGNORE command does not override either stated or automatically INCLUDEd files. Multiple :IGNORE commands and continued commands are allowed.

:DELETE This command causes all files of CONSEC organization for PO tapes or KEYED organization for BO tapes as well as all IGNOREd files to be deleted from the current account as the tape is written. The command has the form

```
:DELETE
```

Table 22. Files Automatically INCLUDEd on BPM BO Tapes[†]

Name	Name
M:MON	M:C
CCI	M:OC
LOADER	M:BI
PASS2	M:CI
PASS1	M:SI
LOCCT	M:BI
PASS3	M:LI
DEF	M:BO
PCL	M:SO
FMGE	M:CO
ERRMSG	M:PO
:DIC	M:GO
:LIB	M:LO
M:DO	M:EO
M:LL	M:CK
M:SL	M:AL

[†] All files are from the :SYS account.

Multiple :DELETE commands are allowed.

The :INCLUDE, :IGNORE, and :DELETE commands apply until the next :WRITE is completed.

:WRITE This command causes DEF to write a tape. The command has the form

```
:WRITE [type][,outsn]
```

where

type specifies a PO or BO tape.

PO specifies that a tape is to be written with the bootable Monitor portion obtained from the current account and all KEYED files obtained from the current account. PO is the default.

BO specifies that a tape is to be written with the bootable Monitor portion obtained from the :SYS account and all CONSEC files obtained from the current account.

outsn specifies the serial number of the tape to be written. If no outsn is specified, the previous outsn for this DCB (M:PO or M:BO) is used again.

Multiple :WRITE commands are allowed.

END This command causes DEF to exit.

If an end-of-file is encountered on input, one of two actions is taken. If the end-of-file follows a WRITE command, DEF exits. Otherwise, DEF writes a PO tape and then exits.

Before DEF is called to write a tape, M:PO and/or M:BO should be ASSIGNed to a device and outsn.

DEF EXAMPLES

1. In the case below, DEF writes a normal BPM PO tape.

```
!ASSIGN M:PO, (DEVICE,9T), (SN,BPM1)
!DEF BPM,F01
!Next Monitor control command
```

2. In the following case, DEF writes LT#BPMO with BLOP among the included items and LT#BPPO with KEYED files K13 and K15 ignored.

```
!ASSIGN M:PO, (DEVICE,9T), (SN,BPM1)
!ASSIGN M:BO, (DEVICE,MT), (SN,UTBO)
!DEF BPM,F01
:INCLUDE (BLOP)
:WRITE BO
:IGNORE (K13,K15)
:WRITE PO,BPPO
!Next Monitor control command
```

Table 23 lists the messages that are output during DEF processing. All messages are output on the LL device.

Table 23. DEF Messages

Message	Description
.....BO TAPE CONTENTS	This is a title message for a list of tape contents.
*****CANNOT OBTAIN 'M:MON' FROM CURRENT ACCOUNT ---------PROCESSOR ABORTED (WRITEMON)	The M:MON load module cannot be obtained from the current account. The processor is aborted.
*****CANNOT OBTAIN 'M:MON' FROM ':SYS' ACCOUNT ---------PROCESSOR ABORTED (WRITEMON)	The M:MON load module cannot be obtained from the :SYS account. The processor is aborted.
****CANNOT OPEN OUTPUT DEVICE	DEF cannot open the output DCB. It proceeds to the next control command.
*****CANNOT READ KEYED RECORD 'HEAD' FROM IN M:MON *****CANNOT READ KEYED RECORD 'MON::ORG' IN M:MON *****CANNOT READ KEYED RECORD 'TREE' IN M:MON *****CANNOT READ KEYED RECORD 'xxxxxxxxxxxx' IN M:MON	One of these messages appears if a part of the M:MON load module cannot be obtained. "xxxxxxxxxxxx' is the name of a M:MON segment. The processor is aborted.
***CANNOT WRITE TAPE	DEF cannot write the tape. It releases the tape and goes to the next control command.
**CC TYPE UNKNOWN ****GET NEXT CC	An unknown control command was encountered. DEF continues.
**DELIMITER MUST BE ',' OR ')'	An invalid delimiter was encountered during the processing of an INCLUDE or IGNORE command. DEF continues to the next command.

Table 23. DEF Messages (cont.)

Message	Description
***ILLEGAL INCLUDE – WILL BE COPIED LATER	An item on an INCLUDE command has the wrong organization for INCLUDE. DEF continues.
******INCLUDE FILE NOT FOUND	An INCLUDE item cannot be obtained. The INCLUDE item name is displayed. DEF continues.
INCLUDE ITEMS	This is a subtitle message for the list of items included on the PO or BO tape.
**NAME INVALID OR > 15 CHAR. LONG	An INCLUDE item name was either too large or not alphanumeric. DEF continues to the next name.
**NO ':' IN COLUMN-1	The control command did not contain a ':' in column 1. (Each control command, including a continuation command, must contain a ':' in column 1.) DEF continues to the next command.
****NOT ENOUGH CORE AVAILABLE ******SYSGEN DEF ABORTED	DEF did not have enough core for its use in processing the INCLUDE or IGNORE command and for writing a PO tape. DEF aborts.
OTHER ITEMS	This is a subtitle message for the list of load modules on the PO tape.
.....PO TAPE CONTENTS.....	This is a title message for a list of the tape contents.
xxxxxxxxxxxx=SEG. #nnnn	This message identifies the segment number (nnnn) for each segment (xxxxxxxxxxxx) as the absolute bootable Monitor is written for the PO tape.
$$$$SEGMENT #'s FOR PATCHING MONITOR MONITOR $$$$	This is a title message that precedes the list of segment numbers.
**SYNTAX ERROR, NO'('	An INCLUDE or IGNORE control command did not define any items. DEF continues to the next command.
:::: SYSGEN DEF IN CONTROL:::: :::: DEF COMPLETED::::	These messages are for title information only.
****WRITING PO BY DEFAULT	DEF is writing a PO tape because of an EOF or as the default of a WRITE command.

CREATION AND USE OF A LOCCT FILE

The following example illustrates how the LOCCT and PASS3 processors may be used. It also shows how a user may generate a LOCCT file for a new processor and then incorporate it into the system. The example is a fictitious one, whose purpose is to show how the System Generation processes may be used. Parts 1-7 comprise one System Generation and parts 8-12 comprise a second System Generation making use of the first System Generation's output.

Example 1:

PART 1

```
!JOB       1378, LOCCTGEN, F
!LOCCT     (LMN, USER), (MAP), (M100), (NOSYSLIB),;
!          (TSS,200), (EF, (AX), (BX), (CX), (DX),;
!          (EX)), (SL, F), (PERM), (BIAS,3000)
!TREE      AX-(BX-(CX, DX), EX)
!DATA
```

Part 1 will result in the LOCCT processor producing a file, LOCCTUSER in account 1378, with the LOCCT table information from the LOAD/TREE command structure. A copy of the file will also be output to the PO device, normally the card punch (this hard copy will be used later in this example in Part 11).

```
:LOCCT      USER
:
.
```

The next and subsequent SYSGENs will not need to have the LOAD/TREE commands to load this element. All that would be needed in PASS3 (Part 11) is a :USER control command.

PART 2

```
!JOB        :SYSGEN, LOCATION, F
!LOCCT      (LMN, COMP), (MAP), (BI)
!DATA
:LOCCT      COMPUTE
:
.
```

Part 2 would result in a LOCCT file generation in the :SYSGEN account with a file-name of LOCCTCOMPUTE. This case will not work correctly when PASS3 references this LOCCT (see Part 11). This is due to the load referencing the BI device. A LOCCT cannot be generated when one of the parameters is BI.

PART 3

```
!JOB        :SYSGEN, PASS1, F
:ASSIGN     M:BI, (SN, BI), (LABEL, X)
!PASS1
:SELECT     (FILE, Z1, Z2, Z3), (FILE, AX, BX, CX, DX, EX)
:SELECT     (FILE, A, B, C, D, E, F, G, H, I, J, K, L, M, N, ;
:           O, P, Q, R, S, T, U, V, W)
:SELECT     (FILE, A1, A2, A3, A4, A5)
:SELECT     (FILE, B1, B2, B3, B4, B5)
:SELECT     (FILE, C1, C2, C3, C4, C5)
:UPDATE     (FILE, X1, X2)

:UPDATE     (FILE, Y1, Y2)
:UPDATE     (FILE, $STD1), (FILE, $STD2), (FILE, $STD3), ;
:           (FILE, #MONRLTMSYM)

!EOD

:LABEL, $STD1
2A1
2A2
2A3
2A4
2A5
!EOD

:LABEL, $STD2
2B1
2B2
2B3
2B4
2B5
!EOD

:LABEL, $STD3
2C1
2C2
2C3
2C4
2C5
!EOD

:LABEL, #MONRLTMSYM
1A
1B
1C
1E
1F
```

Part 3 will select from the BI device (labeled tape) files Z1 through Z3, AX through EX, A through W, A1 through A5, B1 through B5, and C1 through C5. The update function will select from the EI device (card reader) files X1, X2, Y1, Y2, $STD1, $STD2, $STD3, and #MONRLTMSYM. As shown, update files $STD1, $STD2, $STD3, and #MONRLTMSYM are entered to be eventually used as STD files (Standard-System-file). Each record in an STD file contains the file-name of a file which is needed to complete the standard file. These special standard files will be used later in this example (see Part 9).

```
1G
1H
1I
1J
1K
1L
1M
1N
1O
1P
1Q
1R
1S
1T
1U
1V
1W
!EOD
:LABEL, X1
    <deck for X1>
!EOD
:LABEL, X2
    <deck for X2>
!EOD
:LABEL, Y1
    <deck for Y1>
!EOD
:LABEL, Y2
    <deck for Y2>
!EOD
.
.
.
```

PART 4

```
!JOB        :SYSGEN, PASS2, F
!PASS2
:CHAN
:DEVICE     TYA01
:CHAN
:DEVICE     CRA03
:CHAN
:DEVICE     LPA02
:CHAN
:DEVICE     DCAF0, (NSPT,52), (SS, 100), (PER, 0), (PSA, 5),;
:           (PFA, 5F)
:CHAN
:DEVICE     9TA80
:DEVICE     9TA81
:MONITOR    (CORE, 64), (ORG,60), (QUEUE, 20)
:ABS, 1024
:DLIMIT     (TIME, 5)
```

Part 4 contains the PASS2 control commands which describe the target machine. The default set of STDLB operational labels is assumed.

PART 5

```
!JOB        :SYSGEN, LOAD-MONITOR, F
!LOCCT      (LMN, M:MON), (MAP), (NOTCB), (PERM),;
!           (EF, (A), (B), (C), (D), (E), (F), (G), (H), (I),;
!           (J), (K), (L), (M), (N), (O), (P), (Q), (R), (S),;
!           (T), (U), (V), (W)), (BIAS, 0)

!TREE       A-B-C-D-(E-F, G, H-(I, K-L, M),;
!           N-(O-(P, Q), R), S, T-U, V, W)
```

Part 5 is a sample of how the LOCCT tables LOCCTMONRS, LOCCTZZ, LOCCTAA, LOCCTBB, LOCCTCC, LOCCTXX, and LOCCTYY may be generated in the :SYSGEN account. Also, a card punch copy will be generated for each.

```
!DATA
:LOCCT      MONRS
!LOCCT      (LMN, ZZ), (MAP), (M100), (SL, F), (TSS, 80), ;
!           (EF, (Z1), (Z2), (Z3)), (BIAS, 6000), (PERM)
!DATA
:LOCCT      ZZ
!LOCCT      (LMN, AA), (MAP), (M100), (SL, F), (TSS, 0), ;
!           (EF, (A1), (A2), (A3), (A4), (A5)), (PERM), ;
!           (BIAS, 6200)
!DATA
:LOCCT      AA
!LOCCT      (LMN, BB), (MAP), (NOTCB), (PERM), ;
!           (EF, (B1), (B2), (B3), (B4), (B5), (BIAS, 4000)
!DATA
:LOCCT      BB
!LOCCT      (LMN, CC), (MAP), (NOSYSLIB), (PERM), ;
!           (EF, (C1), (C2), (C3), (C4), (C5)), (BIAS, 2600)
!TREE       C1-(C2, C3, C4, C5)
!DATA
:LOCCT      CC
!LOCCT      (LMN, XX), (PERM), (EF, (X1), (X2)), (BIAS, 3000)
!DATA
:LOCCT      XX
!LOCCT      (LMN, YY), (PERM), (EF, (Y1), (Y2)), (BIAS, 3000)
!DATA
:LOCCT      YY
:
:
```

PART 6

```
!JOB        :SYSGEN, PASS1SYSWRT, F
!ASSIGN     M:BO, (DEVICE, 9T), (SN, XBO)
!DEF        , F00
:WRITE      BO
```

Part 6 will generate a new BI tape through M:BO which will contain as ROMs or element files, Z1 through Z3, A through W, A1 through A5, B1 through B5, AX through EX, C1 through C5, X1, X2, Y1, Y2, $STD1, $STD2, $STD3, and #MONRLTMSYM. Also, the LOCCT tables for load module COMP (Part 2), and for M:MON, ZZ, AA, BB, CC, XX, and YY will become part of the new BI tape.

PART 7

```
!JOB        :SYSGEN, DEF, F
!ASSIGN     M:PO, (LABEL, X), (SN, XPO)
!DEF
:INCLUDE    (Z1, Z2, Z3, LOCCTUSER, LOCCTMONRS, ;
:           $STD1)
!FIN
```

Part 7 will generate a PO tape which contains the load modules found in :SYSGEN account and also the element files requested in the DEFs INCL list.

The following parts show how the results of the above System Generation could be used.

PART 8

```
!JOB        :SYSGEN, PASS1, F
!ASSIGN     M:BI, (SN, XBO), (LABEL, X)
!PASS1
:SELECT     (ALL)
```

Part 8 will select every element file from the BI tape (see Part 9 also).

PART 9

```
!JOB        :SYSGEN, PASS, F
!ASSIGN     M:BI, (SN, XBO), (LABEL, X)
!PASS1
```

Part 9 will select through FILE/STD every element file from the BI tape (see Part 8 also).

```
:SELECT     (FILE, Z1, Z2, Z3), (FILE, AX, BX, CX, DX, EX)
:SELECT     (FILE, X1, X2, Y1, Y2)
:SELECT     (STD, $STD1), (STD, $STD2), (STD, $STD3), ;
:           (STD, #MONRLTMSYM)
:SELECT     (FILE, LOCCTCOMPUTE, LOCCTMONRS, ;
:           LOCCTZZ, LOCCTAA, LOCCTBB, LOCCTCC, ;
:           LOCCTXX, LOCCTYY)
```

PART 10

```
!JOB        :SYSGEN, PASS2, F
!PASS2
:CHAN
:DEVICE     TYA01
:CHAN
:DEVICE     CRA03
:CHAN
:DEVICE     LPA02
:CHAN
:DEVICE     DCAF0, (NSPT, 52)(SS, 100), (PER, 0), (PSA, 5);
:           (PFA, 5F)
:CHAN
:DEVICE     9TA80
:DEVICE     9TA81
:MONITOR    (CORE, 64), (ORG, 60), (QUEUE, 20)
:ABS, 1024
:DLIMIT     (TIME, 5)
```

Part 10 is the same as Part 4 above.

PART 11

```
!JOB        :SYSGEN, FMGE, F
!ASSIGN     M:EO, (FILE, LOCCTUSER)
!FMGE       (ENTER, PERM)
    <LOCCT deck from Part 1>
!JOB        :SYSGEN, PASS3, F
!PASS3
:ZZ
:AA         (BIAS=3F09)
:MONRS      (DELETE, SAVE(A, B, C))
:BB         (SAVE(B1), BIAS=+1000)
:CC         (BIAS=2000)
:XX         (DELETE, BIAS=3100)
:YY
:USER       (BIAS=+200)
:COMPUTE
```

Part 11 does essentially the same as Part 5 above, except the first job will enter into the :SYSGEN account the LOCCT table from Part 1.

The command :ZZ will result in loading the load module ZZ, and the bias will remain at 6000 (see original LOAD command), as the load module M:MON does not exist yet.

The command ":AA" will result in loading the load module AA, and the bias will be 4000 (i.e., 3F09 rounded to the next page address). No M:MON load module exists yet.

The command "MONRS" will result in loading the load module M:MON. All element files which are a part of this module will be deleted except for A, B, and C.

The command ":BB" will result in loading the load module BB. The bias will be the first available page address which follows the longest path of the M:MON load module plus 1000. This implies that the bias desired is to be 8 pages above background lower limit. All element files which are a part of this load module will be deleted except for B1.

The command ":CC" will result in loading the load module CC. The bias will be 2000, and if the background lower limit bias determined above for the command :BB is greater than 2000, a warning message will be displayed (e.g., Calculated Background Lower Limit is greater than bias requested).

The command ":XX" will result in loading the load module XX. The bias will be 3200 (i.e., 3100 rounded up to the next page address) and a warning message may be displayed as for the :CC command above. All element files which are a part of this load module will be deleted.

The command ":YY" will result in loading the load module YY. The bias will be the background lower limit as determined for the :BB command above.

The command ":USER" will result in loading the load module USER (from Part 1). The bias will be the address determined for the :BB command above plus 200, one page above background lower limit.

The command ":COMPUTE" will cause the Loader to abort as no element file will be found in the BI file (see Part 2).

PART 12

```
:JOB        :SYSGEN, DEF, F
!ASSIGN     M:PO, (DEVICE, 9T), (SN, XPO)
!DEF
:INCLUDE    (Z1, Z2, Z3, LOCCTUSER, LOCCTMONRS,;
:           $STD1)
:DELETE
!FIN
```

Part 12 will generate a PO tape as Part 7 has defined. All of the element files in the current account will be deleted.

GENERATING STANDARD MONITORS AND PROCESSORS

To simplify the System Generation process, several standard Monitor systems and standard processors have been predefined on the BI tape. This enables an installation to name the particular standard Monitor that is to be generated and automatically have the appropriate elements selected from the BI tape during the execution of PASS1. Also, by referencing the appropriate file name (i.e., specific LOCCT tables) for a standard Monitor and for desired processors, the system will be loaded automatically by PASS3. PASS2 control commands have the form previously described (see "PASS2 Processor").

PASS1 CONTROL COMMANDS FOR STANDARD MONITORS AND PROCESSORS

To perform the standard system selection function in PASS1, it is only necessary to name one of the standard Monitors (see Table 24) and each of the desired processors (see Tables 25 and 26). An example of PASS1 for a minimum BPM system with Meta-Symbol, FORTRAN IV-H, and XDS FORTRAN IV is as follows:

```
!ASSIGN M:BI, (LABEL, X), (SN, BBF1)
!PASS1
:SELECT (STD, $::BPM57M)
!ASSIGN M:BI, (LABEL, X), (SN, SIF1)
!PASS1
:SELECT (STD, $::BFTNH), (STD, $::BPMMETA),;
:    (STD, $::BFTN4)
!EOD
```

Table 24. Standard Monitors

PASS1 Name	LOCCT Name
$::BPM5FDBC	LOCCTBPM5FDBC
$::BPM5RFDBC	LOCCTBPM5RFDBC
$::BPM5SFDBC	LOCCTBPM5SFDBC
$::BPM5SRFDBC	LOCCTBPM5SRFDBC
$::BPM57M	LOCCTBPM57M
$::BPM57R	LOCCTBPM57R
$::BPM57S	LOCCTBPM57S
$::BPM57SR	LOCCTBPM57SR
$::BPM7D	LOCCTBPM7D
$::BPM7SD	LOCCTBPM7SD
$::BTM5SDBC	LOCCTBTM5SDBC
$::BTM5SFDBC	LOCCTBTM5SFDBC
$::BTM57S	LOCCTBTM57S
$::BTM7SD	LOCCTBTM7SD
$::BTMRT5SSIM	LOCCTBTMRT5SSIM
$::BTMRT57S	LOCCTBTMRT57S
$::BTMRT5SDBC	LOCCTBTMRT5SDBC
$::BTMRT7SD	LOCCTBTMRT7SD

See Table 28 for the meaning of the symbols used in PASS1 and LOCCT names.

Table 25. BPM Processors

Processor	PASS1 Name	LOCCT Name
Language		
FORTRAN IV	$::BFTN4	LOCCTBFTN4
		LOCCTBFTN4P (patch version)
FORTRAN IV-H	$::BFTNH	LOCCTBFTNH
FLAG	$::FLAGBO (overlay)	LOCCTFLAGBO
	$::FLAGBN (nonoverlay)	LOCCTFLAGBN
Meta-Symbol	$::BPMMETA	LOCCTBMETA
Symbol	$::BPMSYMBL	LOCCTBPMSYMBL
ANS COBOL	$::BCOBOL	LOCCTBCOBOL
Manage		
Dictionary Processor	$::BDICT	LOCCTBDICT
File Processor	$::BFILP	LOCCTBFILP
Retrieval Processor	$::BRETV	LOCCTBRETV
Report Processor	$::BREPT	LOCCTBREPT
BASIC	$::BPMBASIC	LOCCTBPMBASIC
SL-1	–	(load into any account)
Utility		
Basic Processors (CCI, CCIT, ERRWRT, FMGE, LOADER, LOADERT, OLAY, LOPE)	$::BASEPROC	LOCCTBASEPROC
SYSGEN Processors (PASS1, PASS2, PASS3, LOCCT, DEF)	$::SGENPROC	LOCCTSGENPROC
Standard Utilities (CHKPT, REW, WEOF, PFIL, MONDUMP, MEDDUMP, VOLINIT, PCL)	$::STDUTIL	LOCCTSTDUTIL
Batch Super	$::SUPER	LOCCTSUPER
Special Processors (ELIST, DRFCOM, ROMTRAN, EDCON, FPURGE, FANALYZE)	$::SPECIALS	LOCCTSPECIALS
Application		
Sort-Merge	$::BSORT	LOCCTBSORT
	$::BMERGE	LOCCTBMERGE
DMS: Dump Processor	$::BDMP	LOCCTBDMP
File Definition Processor	$::BFDP	LOCCTBFDP
Initialization Processor	$::BINT	LOCCTBINT
Load Processor	$::BLOD	LOCCTBLOD
Libraries	–	(load into DMS library account)
FMPS	$::FMPS	LOCCTFMPS
GPDS	–	(load into :SYS account)
CIRC	–	(load into any account)
1400 Simulator	$::BSIML	LOCCTBSIML

Table 26. BTM Subsystems

Processor	PASS1 Name	LOCCT Name
Edit	$::EDIT:	LOCCTEDIT:
BASIC	$::BTMBASIC	LOCCTBTMBASIC
FORTRAN IV-H	$::FORTRAN:	LOCCTFORTRAN:
Symbol	$::BTMSYMBL	LOCCTTSYMBL

Table 26. BTM Subsystems (cont.)

Processor	PASS1 Name	LOCCT Name
Load	$::LOAD:	LOCCTLOAD:
Run	$::RUN:	LOCCTRUN:
Delta	$::DELTA:	LOCCTDELTA:
On-line Batch	$::BPM:	LOCCTBPM:
Terminal-Oriented Manage	$::MANAGE:	LOCCTMANAGE:
Ferret	$::FERRET:	LOCCTFERRET:
Super	$::SUPER:	LOCCTSUPER:

For the example system, these are the only control cards necessary for PASS1. Any Monitor or processor in Tables 24 - 26 may be selected in a like manner for other desired systems.

If it is desired to have a complete listing of program names and catalog numbers selected, simply list the standard files from the BI tape using the FMGE processor. If desired, the BI tape can be booted normally and used to run FMGE to list the components of the standard files before generating a system. This is accomplished as follows:

```
!ASSIGN M:EI, (LABEL, $::BPM57M), (SN, BI)
!FMGE (LIST, BCD)
```

PASS3 CONTROL COMMANDS FOR LOADING STANDARD MONITORS AND PROCESSORS

After running PASS1 as described above, and PASS2 as described in a previous section, it is necessary to load the system DCBs (a deck of !LOAD cards for this purpose is furnished with the BI tape). Then, to form the operating system, PASS3 is run. The only PASS3 control card required for loading a Monitor or processor is as shown below.

```
:name [(DELETE[, BIAS=[+]nnnn][, SAVE(n1, n2,...)])]
```

The processor or Monitor to be loaded is identified by "name". The options are needed only to specify the special load bias nnnn (BIAS =nnnn) the load bias offset +nnnn (BIAS = + nnnn), to delete all element files (DELETE), and to save certain element files named n1, n2, etc. (SAVE(n1, n2,...)). The SAVE option automatically implies the deleting of all files that make up "name" except for those specified in the SAVE option.

If element files are to be deleted, it is necessary to use the SAVE option for some of the files, since they must be used more than once. The files that must be saved are shown in Table 27.

Table 27. Element Files that Must Be Saved

Element Files that Must Be Saved	Standard Files in Which They Occur
M:ALDCB	REW, PFIL, WEOF
M:BIDCB	REW, PFIL, WEOF
M:BODCB	REW, PFIL, WEOF
M:CDCB	REW, PFIL, WEOF
M:CIDCB	REW, PFIL, WEOF
M:CKDCB	
M:CODCB	REW, PFIL, WEOF
M:DODCB	REW, PFIL, WEOF
M:EIDCB	PASS3, REW, PFIL, WEOF, FPURGE
M:EODCB	LOCCT, REW, PFIL, WEOF, FPURGE
M:GODCB	REW, PFIL, WEOF
M:JIT	M:MON, CCI
M:LIDCB	REW, PFIL, WEOF
M:LLDCB	REW, PFIL, WEOF
M:LODCB	REW, PFIL, WEOF
M:OCDCB	REW, PFIL, WEOF
M:PODCB	REW, PFIL, WEOF
M:SIDCB	REW, PFIL, WEOF, FPURGE
M:SLDCB	REW, PFIL, WEOF
M:SODCB	REW, PFIL, WEOF, FPURGE
MODIFY	M:MON, PASS3
TAPEFCN	REW, PFIL, WEOF
TPECHST	FMGE, REW, PFIL, WEOF
BPMBT	PASS1, DEF
UTMBPMBT	PASS1, DEF
WRITEMON	

When all standard files containing a given element have been loaded, then that element may be deleted. In general, DCB files should always be saved.

The LOAD and TREE cards for standard Monitors and processors are depicted in Figures 35 through 39.

An example of a complete System Generation for a standard system is shown in Figure 40.

STANDARD MONITORS

All standard Monitors include the handlers for all standard I/O devices except paper tape and will have CALPROC resident.

Any standard Monitor that includes any instruction simulation package will also have ALTCP and MEMALOC resident.

MONITOR TREE STRUCTURE REQUIREMENTS

THE ROOT-RESIDENT MONITOR

The following element files must be in the root. MON::ORG must be first and TOPRT must be last.

MON::ORG
ROOT
M:CPU
M:ABS
ENTRY
IO/IOSYM
[PFSR]
FBCD/DFBCD

[M:SDEV]
[RTROOT]
[M:FRGD]
[DSCIO]
[SIMINT]
[DECSIM]
[FLTSIM]
[BYTSIM]

[CVTSIM]
M:JIT
IOTABLE
HANDLERS
[COOP]
CALPROC
IORT
TOPRT

The following element files must also be in the root for a BTM system:

M:BTM
[BTMSTAT]

When building a new tree structure, the elements currently in the root (e.g., COOP, CALPROC, and IORT) plus any others transferred from the current tree overlays must be within 16383 words below TOPRT. Likewise, any elements in the tree overlays must be within 16383 above TOPRT.

The M:CPU, ENTRY, and M:ABS element files must be in the first 8K of the Monitor root segment.

When generating the Monitor (M:MON load module), it must be loaded by the PASS3/LOCCT process. That is, if a nonstandard Monitor is desired, LOAD/TREE control commands must be processed by the LOCCT processor prior to the execution of PASS3.

Standard Monitor LOCCTs are distributed on the BI tape for user convenience. The LOCCT names for all standard Monitors are made up of a series of codes that are defined in Table 28.

Figure 35 illustrates BPM57M which describes the tree structure for the minimal BPM standard Monitor on the BI tape. This Monitor has no instruction simulators, no symbiont routines, and no real-time.

Figure 36 illustrates BPM5SFDCB which describes the tree structure for a larger BPM standard Monitor. This Monitor has symbiont routines and all instruction simulators.

Figure 37 illustrates BTMRT5SDBC which describes the tree structure for a BTM standard Monitor. This Monitor has time-sharing, real-time, symbiont routines, and all instruction simulators except floating-point.

The tree structures for other standard Monitors are described in the F01-61 release documentation.

Table 28. LOCCT Name Codes

Codes	Description	Additional Root Elements	Root Modifications	Overlay Modifications
BPM	Batch Processing Monitor			
BTM	Batch Time-Sharing Monitor	M:BTM		IOD-BTMNRES replaces IOD.
B	Byte string instruction simulation	SIMINT,BYTSIM		
C	Convert instruction simulation	SIMINT,CVTSIM		
D	Decimal instruction simulation	SIMINT,DECSIM		
F	Floating-point instruction simulation	SIMINT,FLTSIM		
M	Minimal Monitor		DFBCD replaces FBCD. PFSR is not included.	
R or RT	Real-time	M:FRGD,RTROOT		
S	Symbiont	M:SDEV,COOP	IOSYM replaces IO	CCLOSE replaces DUMMYCCL
SIM	All simulators	SIMINT,BYTSIM,CVTSIM, DECSIM,FLTSIM		

```
!LOCCT (LMN,M:MON),;    ***  SIGMA 5/7 BPM MINIMAL ***
!        (BIAS,0),(MAP),(ABS),(SL,F),(PERM),(NOTCB),(NOSYSLIB),(EF,;
!           (MON::ORG),(ROOT),(IOTABLE),(M:ABS),(M:CPU),(M:JIT),;
!           (IORT),(HANDLERS),(TOPRT),(PRGMLDR),(TYPR),(IOD),(DEBUG),;
!           (DUMP),(RDF),(OPNL),(OBSE),(OPN),(CLS),(MODIFY),(EXIT),;
!  (KEYIN2),;
!  (RCVR2),;
!                   (CLS1),(SEGLOAD),(LDPRG),(MEMALOC),(CALPROC),;
!          (WRTF),(WRTD),(LBLT),(POS),(ALTCP),       (M:15),(M:16),;
!          (M:17),(M:18),(M:19),(M:1A),          (M:1E),(PASSOSYS),(MUL),;
!          (IO),(DUMMYCCL),       (DFBCD);
!,(JOBENT),(ENTRY);
!, (RCVR))
!TREE    MON::ORG-ROOT-M:CPU-M:ABS-      ;
! ENTRY-;
!        IO-    DFBCD-;
!        M:JIT-IOTABLE-HANDLERS-CALPROC-IORT-TOPRT-;
!           (PRGMLDR,TYPR,IOD,DEBUG-DUMP,EXIT     ,M:25,M:16,M:17,;
!  KEYIN2,;
!                       M:18,M:1A,JOBENT,        MEMALOC,RDF-;
!               (OPNL-OBSE,M:1E-OBSE,OPN-OBSE,CLS-PASSOSYS-MODIFY-CLS1,;
!MUL,;
!                SEGLOAD-OBSE,WRTF,WRTD-DUMMYCCL,LBLT,M:19 ,POS,ALTCP),;
!  RCVR2,;
! RCVR,LDPRG)
!DATA
:LOCCT    BPM57M
```

Figure 35. BPM57M Standard Monitor

```
!LOCCT (LMN,M:MON),;    ***  SIGMA 5 BPM SYMBIONT-SIMULATORS  ***
!        (BIAS,0),(MAP),(ABS),(SL,F),(PERM),(NOTCB),(NOSYSLIB),(EF,;
!           (MON::ORG),(ROOT),(IOTABLE),(M:ABS),(M:CPU),(M:JIT),;
!           (IORT),(HANDLERS),(TOPRT),(PRGMLDR),(TYPR),(IOD),(DEBUG),;
!           (DUMP),(RDF),(OPNL),(OBSE),(OPN),(CLS),(MODIFY),(EXIT),;
!  (KEYIN2),;
!  (RCVR2),;
!                   (CLS1),(SEGLOAD),(LDPRG),(MEMALOC),(CALPROC),;
!           (WRTF),(WRTD),(LBLT),(POS),(ALTCP),      (M:15),(M:16),;
!           (M:17),(M:18),(M:19),(M:1A),         (M:1E),(PASSOSYS),;
!           (IOSYM),(PFSR),(FBCD),(M:SDEV),(COOP),(CCLOSE),(MUL),;
! (SIMINT),(DECSIM),(FLTSIM),(BYTSIM),(CVTSIM);
!,(JOBENT),(ENTRY);
!, (RCVR))
!TREE    MON::ORG-ROOT-M:CPU-M:ABS-;
! ENTRY-;
!        IOSYM-PFSR-FBCD-M:SDEV-SIMINT-DECSIM-FLTSIM-BYTSIM-CVTSIM-;
!        M:JIT-IOTABLE-HANDLERS-             COOP-CALPROC-IORT-TOPRT-;
!           (PRGMLDR;TYPR,IOD,DEBUG-DUMP,EXIT     ;M:15,M:16,M:17,;
!  KEYIN2,;
!                       M:18,M:1A,MEMALOC,JOBENT,       RDF-;
!               (OPNL-OBSE,M:1E-OBSE,OPN-OBSE,CLS-PASSOSYS-MODIFY-CLS1,MUL,;
!                SEGLOAD-OBSE,WRTF,WRTD-CCLOSE   ,LBLT,M:19 ,POS,ALTCP),;
!  RCVR2,;
! RCVR,LDPRG)
!DATA
:LOCCT    BPM5SFDBC
```

Figure 36. BPM5SFDBC Standard Monitor

```
!LOCCT  (LMN,M:MON),;  *** SIGMA 5 BPM/RT SYMBIONT-DEC/BYT/CVT ***
!        (BIAS,0),(MAP),(ABS),(SL,F),(PERM),(NOTCB),(NOSYSLIB),(EF,;
!           (MON::ORG),(ROOT),(IOTABLE),(M:ABS),(M:CPU),(M:JIT),;
!           (IORT),(HANDLERS),(TOPRT),(PRGMLDR),(TYPR),(IOD),(DEBUG),;
!           (DUMP),(RDF),(OPNL),(OBSE),(OPN),(CLS),(MODIFY),(EXIT),;
!           (RTROOT),(M:FRGD),;
!  (KEYIN1),(KEYIN2),;
!  (RCVR2),;
!                   (CLS1),(SEGLOAD),(LDPRG),(MEMALOC),(CALPROC),;
!           (WRTF),(WRTD),(LBLT),(POS),(ALTCP),      (M:15),(M:16),;
!           (M:17),(M:18),(M:19),(M:1A),        (M:1E),(PASSOSYS),;
!           (IOSYM),(FBCD),(M:SDEV),(COOP),(CCLOSE),(SIMINT),(DECSIM),;
!        (CVTSIM),(BYTSIM),         ;
!(MUL),;
!                   (BTMNRES),(M:BTM),(PFSR);
!,(JOBENT),(ENTRY);
!  ,(RCVR))
!TREE    MON::ORG-ROOT-M:CPU-M:ABS-;
! ENTRY-;
!      SIMINT-DECSIM-       BYTSIM-CVTSIM-;
!        IOSYM-FBCD-M:SDEV-ALTCP-  M:BTM-     PFSR-;
!        RTROOT-M:FRGD- ;
!        M:JIT-IOTABLE-HANDLERS-             COOP-CALPROC-IORT-TOPRT-;
!           (PRGMLDR,TYPR,IOD-BTMNRES,DEBUG-DUMP,EXIT     ,M:15,M:16,M:17,;
!  KEYIN1,KEYIN2,;
!                      M:28,M:1A,MEMALOC,JOBENT,        RDF-;
!               (OPNL-OBSE,M:1E-OBSE,OPN-OBSE,CLS-PASSOSYS-MODIFY-CLS1,;
!MUL,;
!                SEGLOAD-OBSE,WRTF,WRTD-CCLOSE  ,LBLT,M:19,POS),;
!  RCVR2,;
! RCVR,LDPRG)
!DATA
:LOCCT  BTMRT5SDBC
```

Figure 37. BTMRT5SDBC Standard Monitor

```
!LOCCT  (LMN,CCI),(EF,(CCIROOT),(JOB),(LIMIT),(ASSIGN),(LOAD),(TREE);
!       ,(TELSCPE),(RUN),(CCIDBUG),(READBI),(ENDJOB),(ABORT);
!       ,(M:DLIMIT),(M:JIT)),(BIAS,2000),(NOTCB),(MAP),(PERM),(SL,F);
!       ,(NOSYSLIB)
!DATA
:LOCCT  CCI

!LOCCT  (LMN,CCI),(EF,(CCIROOT),(JOB),(LIMIT),(ASSIGN),(LOAD),(TREE);
!       ,(TELSCPE),(RUN),(CCIDBUG),(READBI),(ENDJOB),(ABORT),(M:DLIMIT);
!       ,(M:JIT)),(BIAS,2000),(NOTCB),(MAP),(PERM),(SL,F),(NOSYSLIB)
!TREE   CCIROOT-M:DLIMIT-M:JIT-(JOB,LIMIT,ASSIGN,LOAD,TREE,TELSCPE,RUN;
!       ,CCIDBUG,READBI,ENDJOB,ABORT)
!DATA
:LOCCT  CCIT

!LOCCT  (LMN,FMGE)(EF,(FILEMNGE),(TPECHST),(FMGEDCBS)),(BIAS,2000);
!       ,(MAP),(SL,F),(NOSYSLIB),(PERM),(TSS,100)
!DATA
:LOCCT  FMGE

!LOCCT  (LMN,LOADER),(EF,(LDR),(IN1),(PS1),(IN2),(PS2),(ALL),(EVL);
!       ,(WRT)),(BIAS,2000),(NOTCB),(MAP),(PERM),(SL,F),(NOSYSLIB)
!DATA
:LOCCT  LOADER
```

Figure 38. System Tree Structure

```
!LOCCT  (LMN,LOADER),(EF,(LDR),(IN1),(PS1),(IN2),(PS2),(ALL),(EVL);
!       ,(WRT)),(BIAS,2000),(NOTCB),(MAP),(PERM),(SL,F),(NOSYSLIB)
!TREE   LDR-(IN1,PS1,IN2,PS2-(ALL,EVL,WRT))
!DATA
:LOCCT  LOADERT

!LOCCT  (LMN,OLAY),(EF,(LDR),(IN1),(PS1),(IN2),(PS2),(ALL),(EVL);
!       ,(WRT),,(BIAS,2000),(ABS),(NOTCB),(MAP),PERM),(SL,F);
!       ,(NOSYSLIB)
!TREE   LDR-(IN1,PS1,IN2,PS2-(ALL,EVL,WRT))
!DATA
:LOCCT  OLAY

!LOCCT  (LMN,LOPE),(EF,(LOPEROM)),(BIAS,2000),(MAP),(NOTCB),(PERM)
!DATA
:LOCCT  LOPE

!LOCCT  (LMN,PASS1),(BIAS,2000),(MAP),(SL,F),(TSS,100),(PERM);
!          ,(EF,(PASS1ROM),(P1DCBS),(BPMBT),(UTMBPMBT))
!TREE   PASS1ROM-P1DCBS-(BPMBT,UTMBPMBT)
!DATA
:LOCCT  PASS1

!LOCCT  (LMN,PASS2),(BIAS,2000),(MAP),(SL,F),(TSS,A00),(PERM);
!                        ,(NOSYSLIB);
!         ,(EF,(P2CCI),(P2DCBS),(MODIFY),(UBCHAN),(SDEVICE);
!              ,(XMONITOR),(XLIMIT),(ABS),(FRGD),(BTM),(P2COC);
!              ,(IMC),(SPROCS)
!TREE   P2CCI-P2DCBS-MODIFY-(UBCHAN,SDEVICE,XMONITOR;
!            ,XLIMIT,ABS,FRGD,BTM,P2COC,IMC,SPROCS)
!DATA
:LOCCT  PASS2

!LOCCT  (LMN,PASS3),(BIAS,2000),(MAP),(SL,F),(TSS,200),(PERM);
!          ,(EF,(PASS3ROM),(M:EIDCB),(MODIFY)
!DATA
:LOCCT  PASS3

!LOCCT  (LMN,DEF),(BIAS,2000),(MAP),(SL,F),(TSS,100),(PERM);
!          ,(EF,(DEFROM),(DFDCBS),(BPMBT),(UTMBPMBT))
!TREE   DEFROM-DFDCBS-(BPMBT,UTMBPMBT)
!DATA
:LOCCT  DEF

!LOCCT  (LMN,LOCCT),(BIAS,2000),(MAP),(SL,F),(TSS,100),(PERM);
!          ,(EF,(LOCCTROM),(M:EODCB))
!DATA
:LOCCT  LOCCT

!LOCCT  (LMN,REW),(EF,(TAPEFCN),(TPECHST),(M:OCDCB),(M:CDCB);
!       ,(M:BIDCB),(M:BODCB),(M:SIDCB),(M:EIDCB),(M:LLDCB),(M:LODCB);
!       ,(M:PODCB),(M:EODCB),(M:CODCB),(M:DODCB),(M:CIDCB),(M:SLDCB);
!       ,(M:ALDCB),(M:SODCB),(M:LIDCB),(M:GODCB)),(BIAS,2000);
!       ,(ABS),(MAP),(SL,F),(NOSYSLIB),(PERM),(TSS,100)
!DATA
:LOCCT  REW

!LOCCT  (LMN,WEOF),(EF,(TAPEFCN),(TPECHST),(M:OCDCB),(M:CDCB);
!       ,(M:BIDCB),(M:BODCB),(M:SIDCB),(M:EIDCB),(M:LLDCB),(M:LODCB);
!       ,(M:PODCB),(M:EODCB),(M:CODCB),(M:DODCB),(M:CIDCB),(M:SLDCB);
```

Figure 38. System Tree Structure (cont.)

```
!        ,(M:ALDCB),(M:SODCB),(M:LIDCB),(M:GODCB)),(BIAS,2000);
!        ,(ABS),(MAP),(SL,F),(NOSYSLIB),(PERM),(TSS,100)
!DATA
:LOCCT   WEOF

!LOCCT   (LMN,PFIL),(EF,(TAPEFCN),(TPECHST),(M:OCDCB),(M:CDCB);
!        ,(M:BIDCB),(M:BODCB),(M:SIDCB),(M:EIDCB),(M:LLDCB),(M:LODCB);
!        ,(M:PODCB),(M:EODCB),(M:CODCB),(M:DODCB),(M:CIDCB),(M:SLDCB);
!        ,(M:ALDCB),(M:SODCB),(M:LIDCB),(M:GODCB)),(BIAS,2000);
!        ,(ABS),(MAP),(SL,F),(NOSYSLIB),(PERM),(TSS,100)
!DATA
:LOCCT   PFIL

!LOCCT   (LMN,DEFCOM),(MAP),(SL,F),(EF,(DEFCMROM))
!DATA
:LOCCT   DEFCOM

!LOCCT   (LMN,SUPER),(EF,(SUPERBO),(M:EIDCB),(M:EODCB)),(SL,F),(MAP),(NOTCB);
!        ,(BIAS,2000),(PERM)
!DATA
:LOCCT   SUPER

!LOCCT   (LMN,DELTA:),(EF,(BTMDELTA)),(SL,F),(MAP),(NOTCB),(BIAS,C200)
!DATA
:LOCCT   DELTA:

!LOCCT   (LMN,EDIT:),(EF,(BTMEDIT)),(SL,F),(MAP),(NOTCB),(BIAS,C200);
!        ,(PERM),(ABS)
!DATA
:LOCCT   EDIT:

!LOCCT   (LMN,FERRET:),(EF,(BTMFER)),(SL,F),(MAP),(NOTCB),(BIAS,C200);
!        ,(PERM),(ABS)
!DATA
:LOCCT   FERRET:

!LOCCT   (LMN,LOAD:),(EF,(BTMLOAD)),(SL,F),(MAP),(NOTCB),(BIAS,C200);
!        ,(PERM),(ABS)
!DATA
:LOCCT   LOAD:

!LOCCT   (LMN,BPM:),(EF,(BTMBPM)),(SL,F),(MAP),(NOTCB),(BIAS,C200);
!        ,(PERM),(ABS)
!DATA
:LOCCT   BPM:

!LOCCT   (LMN,SUPER:),(EF,SUPERBO)),(SL,F),(MAP),(NOTCB),(BIAS,C200);
!        ,(PERM)
!DATA
:LOCCT   SUPER:

!LOCCT   (LMN,MONDUMP),(SL,F),(MAP),(PERM),(EF,(MBROOT),(MBSYMTAB),(MBSNAP);
!        ,(MBPSDREG),(MBTRAPS),(MBDCBS),(MBBTMTAB),(MBIOSYM))
!TREE    MBROOT-MBSYMTAB-(MBSNAP,MBPSDREG,MBTRAPS,MBDCBS,MBBTMTAB,MBIOSYM)
!DATA
:LOCCT   MONDUMP

!LOCCT   (LMN,ELIST),(EF,(ELISTROM)),(SL,4),(BIAS,2000),(PERM),(MAP),(ABS)
!DATA
:LOCCT   ELIST
```

Figure 38. System Tree Structure (cont.)

```
!LOCCT  (LMN,FPURGE),(EF,(FPROM),(M:SIDCB),(M:SODCB)),(SL,F),(BIAS,2000),(MAP);
!       ,(PERM),(ABS)
!DATA
:LOCCT  FPURGE

!LOCCT  (LMN,MEDDUMP),(EF,(MEDDBO)),(MAP),(PERM),(BIAS,2000)
!DATA
:LOCCT  MEDDUMP

!LOCCT  (LMN,VOLINIT),(EF,(VOLINBO)),(SL,F),(MAP),(PERM),(BIAS,2000)
!DATA
:LOCCT  VOLINIT

!LOCCT  (LMN,PCL),(MAP),(SL,F),(PERM),(BIAS,2000),(EF,(PCLROM))
!DATA
:LOCCT  PCL

!LOCCT  (LMN,RUN:),(EF,(BTM:RUN)),(SL,F),(MAP),(NOTCB),(BIAS,C200),(PERM),(ABS)
!DATA
:LOCCT  RUN:

!LOCCT  (LMN,FANALYZE),(EF,(FANROM)),(PERM),(BIAS,2000),(SL,F),(TSS,256),(MAP)
!DATA
:LOCCT  FANALYZE

!LOCCT  (LMN,MCHKPT),(EF,(CHKPTROM)),(BIAS,3000),(NOTCB),(MAP),(PERM),(SL,F);
!       ,(NOSYSLIB),(ABS)
!DATA
:LOCCT  CHKPT

!LOCCT  (LMN,EDCON),(EF,(EDCONROM)),(SL,F),(MAP),(NOTCB),(BIAS,2000),(PERM),(ABS)
!DATA
:LOCCT  EDCON

!LOCCT  (LMN,ERRWRT),(EF,(ERROM)),(SL,F),(MAP),(BIAS,2000),(PERM),(ABS)
!DATA
:LOCCT  ERRWRT

!LOCCT  (LMN,ROMTRAN),(EF,(ROMTREN)),(BIAS,2000),(MAP),(NOTCB),(PERM)
!DATA
:LOCCT  ROMTRAN

!LOAD   (LMN,M:CK),(EF,(M:CKDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:AL),(EF,(M:ALDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:BI),(EF,(M:BIDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:BO),(EF,(M:BODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:CI),(EF,(M:CIDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:C ),(EF,(M:CDCB )),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:CO),(EF,(M:CODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:DO),(EF,(M:DODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:EI),(EF,(M:EIDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:GO),(EF,(M:GODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:EO),(EF,(M:EODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:LI),(EF,(M:LIDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:LL),(EF,(M:LLDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:LO),(EF,(M:LODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:OC),(EF,(M:OCDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:PO),(EF,(M:PODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:SI),(EF,(M:SIDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD   (LMN,M:SL),(EF,(M:SLDCB)),(MAP),(PERM,LIB),(NOSYSLIB)
```

Figure 38. System Tree Structure (cont.)

```
!LOAD    (LMN,M:SO),(EF,(M:SODCB)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD    (LMN,SSS ),(EF,(SSSROM )),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD    (LMN,S:OVERLY),(EF,(S:OVERLYR)),(MAP),(PERM,LIB),(NOSYSLIB)
```

Figure 38. System Tree Structure (cont.)

```
!LOCCT   (LMN,FORTRAN),(NOSYSLIB),(MAP),(ABS),(SL,F),(PERM),(SEQ),(NOTCB);
!        ,(EF,(M:BO,:SYS),(M:C,:SYS),(M:GO,:SYS),(M:LO,:SYS),(M:OC,:SYS);
!        ,(M:SI,:SYS),(M:SO,:SYS),(M:DO,:SYS),(PASSOF),(MERRIMAC),(FORTRAN4);
!        ,(INTERPTR),(LISTDEFS),(FORTTEMP),(X1X2DCBS),(PASS1F),(STASCAN),(EXSCAN);
!        ,(CONSCAN),(PASS1ML),(SINCOLM1),(P2P3ROOT),(PASS2F),(ALLOCATE),(PASS3F);
!        ,(GENLIT),(OUTPROG),(OUTBO),(PRINTCON),(SUMMARY))
!TREE    PASSOF-MERRIMAC-FORTRAN4-INTERPTR-LISTDEFS-FORTTEMP-X1X2DCBS-;
!        ,M:BO,M:C-M:GO-M:LO-M:OC-M:SI-M:SO-M:DO-;
!        ,(PASS1F-STASCAN-EXSCAN-CONSCAN-PASS1ML-SINCOLM1,P2P3ROOT-,(PASS2F-ALLOCATE;
!        ,PASS3F-GENLIT-OUTPROG-OUTBO-PRINTCON-SUMMARY))
!DATA
:LOCCT   BFTN4
!LOCCT   (LMN,FORTRANH),(EF,(BFTNH),(M:SI,:SYS),(M:LO,:SYS),(M:LO,:SYS),(M:BO,:SYS);
!        ,(M:GO,:SYS),(M:DO,:SYS),(M:LL,:SYS),(M:C,:SYS)),(NOSYSLIB),(MAP),(ABS);
!        ,(SL,F),(PERM),(SEG),(BIAS,5000)
!DATA
:LOCCT   BFTNH
!LOCCT   (LMN,SORT),(MAP),(PERM),(ABS),(SEG),(ERSTACK,0),(ERTABLE,0),(BIAS,5000);
!        ,(EF,(S:SORTD),(S:SORTPO),(S:SORTP1),(S:SORTP2),(S:SORTP3),(S:DCB1)
!TREE    S:DCB1-S:SORTP-(S:SORTP1-S:SORTP2-S:SORTP3)
!DATA
:LOCCT   BSORT
!LOCCT   (LMN,MERGE),(EF,(MERGEO)),(PERM),(MAP),(ABS),(BIAS,5000),(TSS,0),(SL,F)
!DATA
:LOCCT   BMERGE
!LOCCT   (LMN,SIML),(EF,(SIML)),(BIAS,5000),(MAP),(PERM),(SL,F),(ABS),(TSS,100)
!DATA
:LOCCT   1401SIM
!LOCCT   (LMN,TCON),(EF,(TAPECON)),(BIAS,5000),(MAP),(PERM),(SL,F),(ABS)
!DATA
:LOCCT   1401TCON
!LOCCT   (LMN,BASIC:),(EF,(OBASIC)),(SL,F),(MAP),(NOTCB),(BIAS,C200),(PERM),(ABS)
!DATA
:LOCCT   BTMBASIC
!LOCCT   (LMN,BASIC),(EF,(BBASIC),(M:CIDCB),(M:DODCB),(M:EIDCB),(M:EODCB),(M:LODCB);
!        ,(M:SIDCB),(M:SODCB)),(SL,F),(MAP),(NOTCB),(BIAS,5000),(PERM),(ABS)
!DATA
:LOCCT   BPMBASIC
!LOCCT   (LMN,FORTRAN:),(EF,(BTMFINT),(BTMFORT)),(SL,F),(BIAS,C200),(MAP),(NOTCB);
!        ,(PERM),(ABS)
!DATA
:LOCCT   FORTRAN:
!LOCCT   (LMN,METASYM),(MAP),(PERM),(ABS),(BIAS,5000),(SL,F),(TSS,1A),(EF,(BINTRPR);
!        ,(BEML),(BEPL),(BCC1),(BCONC),(BINTTBL),(BAML),(BAPL),(BOPR),(BOBJ);
!        ,(M:C,:SYS),(M:BO,:SYS),(M:GO,:SYS),(M:LO,:SYS),(M:SO,:SYS),(M:CI,:SYS);
!        ,(M:SI,:SYS),(M:CO,:SYS),(M:DO,:SYS))
!TREE    BINTRPR-;
!        ,M:C-M:BO-M:GO-M:LO-M:CI-M:CO-M:SI-M:SO-M:DO-;
!        (BEML-BEPL-BCCI-BCONC-BINTTBL,BAML-BAPL-BOPR-BOBJ)
!DATA
:LOCCT   BMETA
!LOCCT   (LMN,SYMBOL),(EF,(BI704159)),(BIAS,5000),(PERM),(MAP),(SL,F),(ABS)
```

Figure 39. Language Processor Load/Tree Structure

```
!DATA
:LOCCT  BSYMBL
!LOCCT  (LMN,SYMBOL:),(EF,(BI705399)),(SL,F),(MAP),(NOTCB),(BIAS,C200),(PERM),(ABS)
!DATA
:LOCCT  TSYMBL
```

Figure 39. Language Processor Load/Tree Structure (cont.)

```
*
:GENDCB     (M:BI,:SYSGEN,(INSN,BBF1),9T)
!END
!JOB        :SYSGEN,DO$SYSGEN,F
!LIMIT      (TIME,60)
!ASSIGN     M:BI,(LABEL,X),(SN,BBF1)
!PASS1
:SELECT     (STD,$::BPM),(STD,$::BASEPROC),(STD$::SGENPROC),(STD,$::STDUTIL);
:           ,(STD,$::SPECIALS)
!ASSIGN     M:BI,(LABEL,X),(SN,SIF1)
!PASS1
:SELECT     (STD,$::BPMMETA),(STD,$::BPMSYMBL)
!PASS2
:CHAN
:DEVICE     TYA01
:CHAN
:DEVICE     CRA03
:CHAN
:DEVICE     LPA02
:CHAN
:DEVICE     9TA80
:DEVICE     9TA81
:CHAN
:DEVICE     DCAF0,(7204),(PFA,1D0),(PSA,30),(PER,0)
:MONITOR    (CORE,64),(TSTACK,300),(CFU,11),(QUEUE,16),(ORG,60)
:ABS,1024   (CCI)
:DLIMIT     (TIME,5),(LO,250),(PO,100),(DO,250),(UO,250)
!PASS3
:BP         (SAVE(M:JIT,MODIFY),BIAS=0)
:CC1        (DELETE)
:PASS2      (SAVE(MODIFY))
:LOADER
:OLAY       (DELETE)
:PASS1
:LOCCT
:PASS3      (SAVE(M:EIDCB))
:DEF
:LOPE       (DELETE)
:METASYM    (SAVE(M:CDCB,M:BODCB,M:GODCB,M:LODCB,M:CIDCB,M:CODCB,M:SIDCB,M:SODCB))
:SYMBOL     (SAVE(M:OCDCB,M:DODCB,M:GODCB,M:SIDCB,M:BODCB,M:LODCB,M:CDCB))
:FMGE       (SAVE(TPECHST))
:REW        (SAVE(TAPEFCN,TPECHST,M:OCDCB,M:CDCB,M:BIDCB,M:BODCB,M:SIDCB,M:EIDCB;
:           ,M:LLDCB,M:LODCB,M:PODCB,M:EODCB,M:CODCB,M:DODCB,M:CIDCB,M:SLDCB;
:           ,M:ALDCB,M:SODCB,M:LIDCB,M:GODCB))
:PFIL       (SAVE(TAPEFCN,TPECHST,M:OCDCB,M:CDCB,M:BIDCB,M:BODCB,M:SIDCB,M:EIDCB;
:           ,M:LLDCB,M:LODCB,M:PODCB,M:EODCB,M:CODCB,M:DODCB,M:CIDCB,M:SLDCB;
:           ,M:ALDCB,M:SODCB,M:LIDCB,M:GODCB))
                                      {PASS3 IS FOLLOWED BY
                                      {!LOAD CONTROL COMMANDS FOR
                                      {MONITOR DCB'S, SSS, AND
                                      {S:OVRLY
```

Figure 40. System Generation Example

```
!ASSIGN    M:PO,(DEVICE,9T),(SN,PO)
!DEF
:INCLUDE   (BPM,MON,SIG7FDP,SIGMET)
!FIN
```

Figure 40. System Generation Example (cont.)

HORIZONTAL TREE INSTRUCTIONS

RDF cannot be in the same overlay area with any of the following:

CLS
LBLT
M:19
M:1E
MUL
OPN
OPNL
POS
SEGLOAD
WRTF

VERTICAL TREE INSTRUCTIONS

Within any overlay area the vertical order of the standard tree structure must not be changed.

NUMBER OF OVERLAY LEVELS

The standard tree structure has two levels. The maximum allowed is four levels (unless MAXLEV is changed).

OVERLAY REQUIREMENTS

WRTD and CCLOSE/DUMMYCCL must be in the same segment. CLC, PASS0SYS, MODIFY, and CLS1 must be in same segment with the order unchanged and also must be in the last level of the tree structure.

DEBUG and DUMP, OPNL and OBSE, M:1E and OBSE, OPN and OBSE, SEGLOAD and OBSE must be as stated, unless DUMP and/or OBSE is put into the root of the Monitor.

BTMNRES must be included in the same segment with IOD in a BTM system.

USER INITIALIZATION ROUTINES

The following user initializer routines (supplied by the user) may be incorporated into the Monitor tree structures as shown below. Alternatively, the routine may be incorporated in the Monitor root segment.

USRINIT1 USRINIT2 }	Following TYPR in the TYPR segment (i.e., TYPR-USRINIT1-USRINIT2)
USNRINIT1 USNRINIT2 }	In CLS segment (i.e., CLS – USNRINIT1 – USNRINIT2 – PASS0SYS – MODIFY – CLS1)

BTM SYSTEM GENERATION

This discussion is intended to supplement the preceding SYSGEN discussion for users who have a BTM system.

The BI tape for BTM must have all the normal BPM modules plus the following files:

File	Contents (ROMs)
COC	BTM power on/off, initialization, and Executive routines.
BTM:BLIB	On-line FORTRAN run-time and math library.
BTMFINT	On-line FORTRAN interface program.
BTMFORT	On-line FORTRAN compiler (cat. no. 704176).
BTMLOAD	LOAD subsystem (cat. no. 705260).
BTMEDIT	EDIT subsystem.
BTMSYMB	On-line SYMBOL (cat. no. 704158).
BTMFER	FERRET subsystem.
BTMBPM	BPM subsystem.

File	Contents (ROMs)
BTMBASIC	On-line BASIC.
BTM	SYSGEN :BTM card interpreter.
BTMRUN	RUN subsystem (cat. no. 705698).

SYSGEN OPERATIONAL INFORMATION

PASS 1

The :SELECT cards should select the additional modules needed for the particular system being generated:

COC is required for any BTM system.

BTM:BLIB is required for FORTRAN.

BTM is required only if a PASS2 processor is to be part of the object system.

Note: BASIC and FORTRAN require floating-point, so the appropriate simulator must be included in the absence of the hardware option.

PASS 2

:STDLB and :DEVICE cards should be set up as in BPM except as discussed below.

The last n RADs in the HGP chain are used for swapping. Swapping space is allocated from the last RAD in the HGP chain first, the next to last RAD next, etc. The last allocated RAD, and only the last one, may have PFA or PER space allocated on it as well. Note that the last n RADs, assuming n swappers, must have the same sector size and the same number of sectors per track. During the PO tape boot, if there is not sufficient space on available RADs for all users, the following message will be typed on the operator's console:

```
NO ROOM FOR USERS, CAN'T HAVE BTM
```

BTM will be disabled but BPM will still run.

The amount of subsystem swap storage required may be computed as the sum of the following items that are used:

Subsystem	Storage
FORTRAN	34 granules
LOAD	11 granules
EDIT	10 granules
SYMBOL	14 granules
FERRET	6 granules
BPM	4 granules
BASIC	20 granules

User swap area, in granules, may be computed as

(2*(USERSIZE/512) + 4) * (NUMUSERS)

The total swap area required is the sum of subsystem and user swap areas.

For example, 680 granules are needed for 10 users with a 16K on-line area. The definition of a granule is the BPM definition, i.e., that number of sectors needed to make up one page (512 words). If all of the subsystems listed above are used, the total swap area required is 779 granules.

If the swap areas are to be allocated on a disk pack, each user swap area is allocated in whole-cylinder increments; during the SYSGEN process, adjustment of values for PSA, PFA, and PER may be required if they reside on the same device.

The allocation of swap area can be checked after the system is booted. The system types out

```
BTM SWAP AREA IS FROM aaaa TO END OF DISC ndd.
```

An installation desiring to add subsystems subsequent to SYSGEN should allow ample USERSIZE when generating the system. If the swapping area is to be shared with system file storage, system file storage will be wiped out if enough space is not allowed.

A card defining the communications controller address should be included as follows:

:DEVICE COndd, (HAND, COC, COC)

:SDEVICE, :DLIMIT, and :ABS cards are identical to those used with normal BPM.

:MONITOR card is identical. However, the following recommendations are made:

TSTACK, 300 for symbiont system (250 for nonsymbiont).

SPOOL, CPOOL, MPOOL, SFIL, no change.

CORE The correct size must be specified; the system cannot be patched to run on a different size machine.

QUEUE, 20

CFU If none is specified, 10 are supplied. With BTM, the number should be NUMUSER * 2 + 10.

A :BTM card must be included to generate a BTM system. It generates tables that are dependent upon user area size, number of users, etc. It may appear anywhere after the :DEVICE and :SDEVICE cards.

The interpretation of the :BTM card produces a load module named M:BTM, which contains all of the variable storage needed by the resident Executive. The resident Executive (BTMBO) and M:BTM are included in the generation of the Monitor.

BTM is designed to run under Clock3 (500 Hz). Absolute core location X'54' contains the counter 3 count pulse when a user is running.

```
    MTW, -1 TSDTIMER
```

TSDTIMER is refreshed to the value BTMQTM every time batch gives up control, where BTMQTM is supplied by either the KEYIN routine or :BTMCCI. Obviously BTMQTM must be divided by 2 (by the BTM Executive) in order to ensure the correct millisecond value desired to satisfy the 500 Hz pulse. This is done. Furthermore, it is not suggested that a BTM system be run under another rate Clock3 because of complications in timing resolutions. A clock setting faster than 500 Hz will occasionally cause TSDTIMER to go negative. Any setting other than 500 Hz will make timing data incorrect.

LOAD AND OVERLAY CARDS

BPM processor control cards are unchanged.

The module name and element file entries have the following values for each of the subsystems:

Subsystem	LMN	EF(s)
FORTRAN	FORTRAN:	BTMFINT, BTMFORT
LOAD	LOAD:	BTMLOAD
EDIT	EDIT:	BTMEDIT
SYMBOL	SYMBOL:	BTMSYMB
SUPER	SUPER:	BTMSUPER
FERRET	FERRET:	BTMFER
BPM	BPM:	BTMBPM
BASIC	BASIC:	BTMBASIC
RUN	RUN:	BTMRUN

DEF CARD

The format of the DEF card is exactly as in BPM. If on-line FORTRAN is to be used, BTM:BLIB should be specified in the (INCL,...) list.

SYSTEM BOOT AND INITIALIZATION

System boot and initialization is exactly as in BPM.

JOB COMMAND

After the system has been generated, the first of the following two jobs must be run if on-line FORTRAN is to be used. The second job is optional but is recommended, as it maximizes available disk space.

```
!JOB :BTM, ONLINELIB, F                  } Creates :BLIB
!ASSIGN M:BI, (FILE, BTM:BLIB, :SYS)     } file under
!LOPE (PERM, LIB)                        } account :BTM
!JOB :SYS, DELETE                        } Deletes
!ASSIGN M:EI, (FILE, BTM:BLIB)           } BTM:BLIB
:FMGE (DELETE)                           } in :SYS
```

The :BLIB file in the :BTM account is used as the standard library file by the LOAD subsystem.

COMPONENT SIZES IN A BTM SYSTEM

FIXED OVERHEAD

Fixed overhead is as follows:

BPM (F00)	7K (Nonsymbiont) - 10K (Symbiont)
BTM Exec.	4.5K

VARIABLE OVERHEAD

Tables are generated at SYSGEN time which vary with the number of users and subsystems. The formula for the number of words required is:

$$(NU(IB+OB)+3)/4+(9NS+8)/2+23(NU+1)+(5(5NU+12))/4+(3(NS+2))/4$$

where

IB = input buffer size in characters

OB = output buffer size in characters

NU = number of users

NS = number of subsystems

Figures for three typical systems are:

IB = 100, OB = 100, NS = 12

NU = 8	NU = 16	NU = 24
1K(735 words)	1.5K(1369 words)	2K(2003 words)

For safety, the numbers are rounded up to the nearest page, then combined with the fixed overhead figures. Therefore, resident requirements for an 8-user symbiont system are 17K.

BACKGROUND AND ON-LINE AREAS

These memory areas must be allocated from the remainder.

on-line = USERSIZE

background = (core size less the sum of on-line area, context area, and monitor size).

2K of context area is allocated from the background area. Background Monitor blocking buffers, normally 1.5K - 2K, are allocated from the background area. On-line blocking buffers are contained in the context area.

SYSGEN DECK SETUP

The listing in Figure 41 shows the System Generation deck setup for a representative BTM system. This deck generates a 32-user, 80K symbiont system.

```
*
:GENDCB  (M:BI,:SYSGEN,(INSN,BBF1),9T)
!END
!JOB :SYSGEN,PASS1,F
!LIMIT (TIME,9999),(LO,9999),(PO,9999),(TSTORE,9999),(PSTORE,9999)
!ASSIGN M:BI,(LABEL,X),(SN,BBF1)
!PASS1 BPM
:SELECT (ALL)
!ASSIGN M:EI,(LABEL,X),(SN,SIF1)
:UPDATE (STD,$::BPMMETA),(STD,$::BPMSYMBL)
!EOD
!JOB :SYSGEN,PASS2,F
!LIMIT (TIME,9999),(LO,9999),(PO,9999),(TSTORE,9999),(PSTORE,9999)
!PASS2 BPM
:CHAN
:DEVICE  TYA01
:CHAN
:DEVICE  CRA03
:CHAN
:DEVICE  LPA02
:CHAN
:DEVICE  CPA04
:CHAN
:DEVICE  9TA80
:DEVICE  9TA81
:DEVICE  9TA82
:DEVICE  9TA83
:CHAN
:DEVICE  7TAEC
:DEVICE  7TAE1
:CHAN
:DEVICE DCCF0,(HAND,DISCIO,DISCCU),(SS,100),(NSPT,C),(PSA,50),(PFA,100),(PER,B0)
:DEVICE DCCF1,(HAND,DISCIO,DISCCU),(SS,100),(NSPT,C),(PSA,00),(PFA,200),(PER,00)
:DEVICE DCCF2,(HAND,DISCIO,DISCCU),(SS,100),(NSPT,C),(PSA,00),(PFA,200),(PER,00)
:DEVICE DCCF3,(HAND,DISCIO,DISCCU),(SS,100),(NSPT,C),(PSA,00),(PFA,200),(PER,00)
:CHAN
:DEVICE DPD80,(HAND,DPAK,DPAKCU),(SS,100),(NSPT,6),(PFA,FA0),(PER,0),(PSA,0)
:DEVICE DPD81,(HAND,DPAK,DPAKCU),(SS,100),(NSPT,6),(PFA,FA0),(PER,0),(PSA,0)
:CHAN
:DEVICE DCBF0,(HAND,DISCIO,DISCCU),(SS,100),(NSPT,52),(PFA,0),(PER,0),(PSA,0)
:CHAN
:DEVICE COA10,(HAND,COC,COC)
:SDEVICE (LMN,ISSEG,CRA03),(LMN,OSSEG,LPA02,CPA04)
:MONITOR (TSTACK,350),(CORE,80),(MPATCH,50),(QUEUE,20),(SPOOL,8),(MPOOL,10);
:         ,(CPOOL,6),(ORG,62),(SFIL,60),(CFU,75)
:DLIMIT     (TIME,99),(LO,9999),(PO,9999),(DO,9999),(UO,9999),(TSTORE,4096);
:            ,(PSTORE,4096),(IPOOL,4),(FPOOL,4)
:ABS,1C24              (CCI),(METASYM),(FMGE),(LOPE)
:BTM (NUMUSERS,32),(USERSIZE,18432)
```

Figure 41. 80K System Generation Example

```
!JOB :SYSGEN,LOADDCB,F
!LIMIT (TIME,9999),(LO,9999),(PO,9999),(TSTORE,9999),(PSTORE,9999)
!LOAD (LMN,M:CK),(EF,(M:CKDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:AL),(EF,(M:ALDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:BT),(EF,(M:BIDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:BO),(EF,(M:BODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:CI),(EF,(M:CIDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:C ),(EF,(M:CDCB ,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:CO),(EF,(M:CODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:DO),(EF,(M:DODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:EI),(EF,(M:EIDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:GO),(EF,(M:GODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:EO),(EF,(M:EODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:LI),(EF,(M:LIDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:LL),(EF,(M:LLDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:LO),(EF,(M:LODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:OC),(EF,(M:OCDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:PO),(EF,(M:PODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:SI),(EF,(M:SIDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:SL),(EF,(M:SLDCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,M:SO),(EF,(M:SODCB,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,SSS),(EF,(SSSROM,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!LOAD (LMN,BDB),(EF,(BDBROM,:SYSGEN)),(MAP),(PERM,LIB),(NOSYSLIB)
!JOB :SYSGEN,PASS3,F
!LIMIT (TIME,9999),(LO,9999),(PO,9999),(TSTORE,9999),(PSTORE,9999)
!PASS3  BTM
:BTM57S     (BIAS=0)
:CCI
:FMGE
:DEF
:LOADERT
:LOCCT
:PASS1
:PASS2T
:PASS3
:PFIL
:REW
:WEOF
:LOPE
:SUPER
:SYMBOL
:BASIC:     (BIAS=F800)
:BPM:       (BIAS=F800)
:DELTA:     (BIAS=F800)
:EDIT:      (BIAS=F800)
:FERRET:    (BIAS=F800)
:FORTRAN:      (BIAS=F800)
:LOAD:         (BIAS=F800)
:METASYM
:ELIST
:FPURGE
:SORT
:RUN:          (BIAS=F8C0)
:SUPER:     (BIAS=F800)
:SYMBOL:       (BIAS=F8C0)
!JOB :SYSGEN,DEF,F
!LIMIT (TIME,9999),(LO,9999),(PO,9999),(TSTORE,9999),(PSTORE,9999)
!POOL (FPOOL,5),(IPOOL,5)
!ASSIGN M:PO,(DEVICE,9T),(OUTSN,BTRC)
!DEF BPM
:INCLUDE (SIG7FDP,BPM,:BLIB,BTM:BLIB,MON)
FIN
```

Figure 41. 80K System Generation Example (cont.)

REAL-TIME SYSTEM GENERATION

Whenever a real-time system is generated, the tree structure of the Monitor LOCCT table should be organized so that the ALTCP and CALPROC segments are placed in the root of the Monitor. This is necessary because of the manner in which an exit from a centrally connected foreground task is performed.

During the processing of the M:EXIT procedure, control is transferred to routines contained in the ALTCP and CALPROC segments. If these segments are not contained either in the root of the Monitor or resident in core, they must be loaded from the disk. The time required to access and read the disk must be added to the maximum response time (approximately 400 microseconds) that must elapse before a centrally connected task may be entered again. Depending on the number of disk transfers involved, this elapsed time could easily approach 120 milliseconds.

If a user decides that his foreground programs cannot operate within the time restraints mentioned above (i.e., 400 microsecond response time), he will have to connect his foreground tasks directly to an interrupt. Directly connected foreground tasks are given control immediately upon occurrence of an interrupt. Centrally connected tasks first give control to a Monitor routine that saves the system environment and switches environment to the foreground task.

Allocation of contiguous disk storage may be done at SYSGEN time or at run-time. Contiguous disk storage can be addressed by relative sector.

The following commands have options that apply to the generation of real-time systems:

:STDLB

:INTLB

:DEVICE

:MONITOR

:FRGD

These commands are discussed below only to the extent that they concern real-time users.

:STDLB All standard Monitor operational labels are defined by means of the :STDLB control command.

The :STDLB command has the form

:STDLB (label, name)[, (...), ...]

where

label specifies a Monitor or foreground operational label. All operational labels must consist of one or two alphanumeric characters.

name specifies a physical device name to which the operational label is to be assigned.

Foreground operational label assignments may be changed only through the unsolicited key-in !SYST.

:INTLB This control command provides capability of associating a label with an interrupt location. The label may then be used in the different interrupt CALs of the Monitor such as M:ARM. This command has the form

:INTLB (label, loc)[, (...)...]

where

label specifies a label consisting of one or two alphanumeric characters.

loc specifies the absolute hexadecimal interrupt location to be associated with the label.

This control command must immediately follow the :FRGD command. The unsolicited key-in !INTLB may be used to change the assignment of the label from one interrupt location to another.

:DEVICE This control command is used to introduce peripheral units and their handlers into the Monitor system and to describe the attributes peculiar to the device. The command has the form

:DEVICE name[, number][, (option)]...

The following options apply to real-time direct-access disk storage:

BCHK, value specifies the hexadecimal number of tracks to be allocated for checkpoint of the background area by a foreground task. The size of this area must be large enough to hold the entire background area.

ABSF, value specifies the hexadecimal number of tracks to be allocated for absolute foreground programs formed by specifying the ABS option in the !RUN control command or via the INTS option on the :FRGD command. This space must be large enough to hold all absolute foreground programs. Programs can be deleted from the area or packed by the !ABSF key-in.

:MONITOR When a real-time system that uses external interrupts is generated, the Monitor must be BIASed above the highest external interrupt used. This is accomplished by use of the ORG option on the :MONITOR command. The form of the command is

:MONITOR[(option)]

where

ORG, value specifies the absolute hexadecimal BIAS of the Monitor. If this option is omitted from the :MONITOR command, a default of X'60' is used.

FQUEUE, value   specifies, in decimal, the number of queue entries reserved for foreground programs. The value specified must be less than the number of entries defined for QUEUE (see above). The default is 0 if omitted (or incorrect).

FMPOOL, value   specifies, in decimal, the number of Monitor buffers reserved for foreground use. The value specified must be less than the number specified for MPOOL (see above). The default is 0 if omitted (or incorrect).

TSTACK, size   This option is used to specify the number of words in Monitor's temp stack. The default is 200 for nonsymbiont systems and 250 for symbiont systems. The Monitor saves its environment as well as the TEMP area during interrupt processing. Thus, the more interrupts there are, the more TSTACK space is required. It is recommended that the user supply an additional 50 words for each interrupt level.

:FRGD   This command is used to define the structure of a real-time system in terms of memory layout, number of foregrounds in the system, number of interrupts to be used, Monitor's control task interrupt, etc. This command must be used in generating a real-time system. The format of this command is

:FRGD (option)

where

NFRGD, size   specifies the maximum number of foreground programs known to the Monitor at one time. This includes foreground programs which are absolute on the disk, resident in memory, or both.

CT, address   specifies the absolute hexadecimal interrupt location to which the Monitor's control task is to be connected.

FCOM, size   specifies the (decimal) number of pages of foreground COMMON. The default size is zero.

RESDF, size   specifies the (decimal) number of pages to be reserved for foreground programs. This space is available for all foreground programs, regardless of how they are loaded, on a first-come, first-serve basis. A program is given the necessary space when it is loaded for execution and no other program may use the space until the executing program releases it. The default size is zero.

FIPOOL, value   specifies the number of file index buffers to be allocated for foreground programs. The default value is zero.

FFPOOL, value   specifies the number of file blocking buffers to be allocated for foreground programs. The default value is zero.

NINT, value   specifies the maximum number of interrupts which will be used at one time. This includes the clock and external interrupts but does not include the external interrupt for the Monitor's control task.

INTS, ($name_1$, $size_1$, $page_1$)...   specifies the names of foreground programs which are to be made absolute and loaded as resident foreground programs when the Monitor is booted from a PO or BI tape. These programs are also loaded when the Monitor is booted from disk. The decimal value "size" specifies the additional number of words to be allocated on disk for expansion if the program is updated. The decimal value "page" specifies the additional number of pages to be allocated for the program when it is loaded into memory. The load modules for these programs must have been formed at System Generation time by the reloading loader. Also, when they are loaded they are given control at their end-transfer address so that they can initialize themselves.

CTQ, value   specifies the number of entries in the control task queue. These entries are for queuing up M:SBACK and M:ABSLOAD requests.

All file index and blocking buffers for foreground reside at the high end of core or immediately precedes the BTM area in a BTM system. The two foreground areas FCOM and RESDF precede these buffers. The background area always follows the Monitor and the size of the background area is the amount of space left after all other areas have been allocated.

A layout of memory in a real-time system is shown in Figure 42.

REMOTE BATCH SYSGEN

During PASS2 the Data Set Controllers (DSC) for the Remote Batch Terminal (RBT) must be specified in the same way that any other multidevice controller is specified. That is,

- Each RBT included in the system must be defined by a set of :CHAN, :DEVICE, and :SDEVICE commands.

- There must be no more than one :CHAN command per controller.

- Immediately following the :CHAN, there must be three :DEVICE commands which may use any of the standard options associated with unit record devices. The three :DEVICE commands must specify a card reader, a printer, and a card punch as devices.

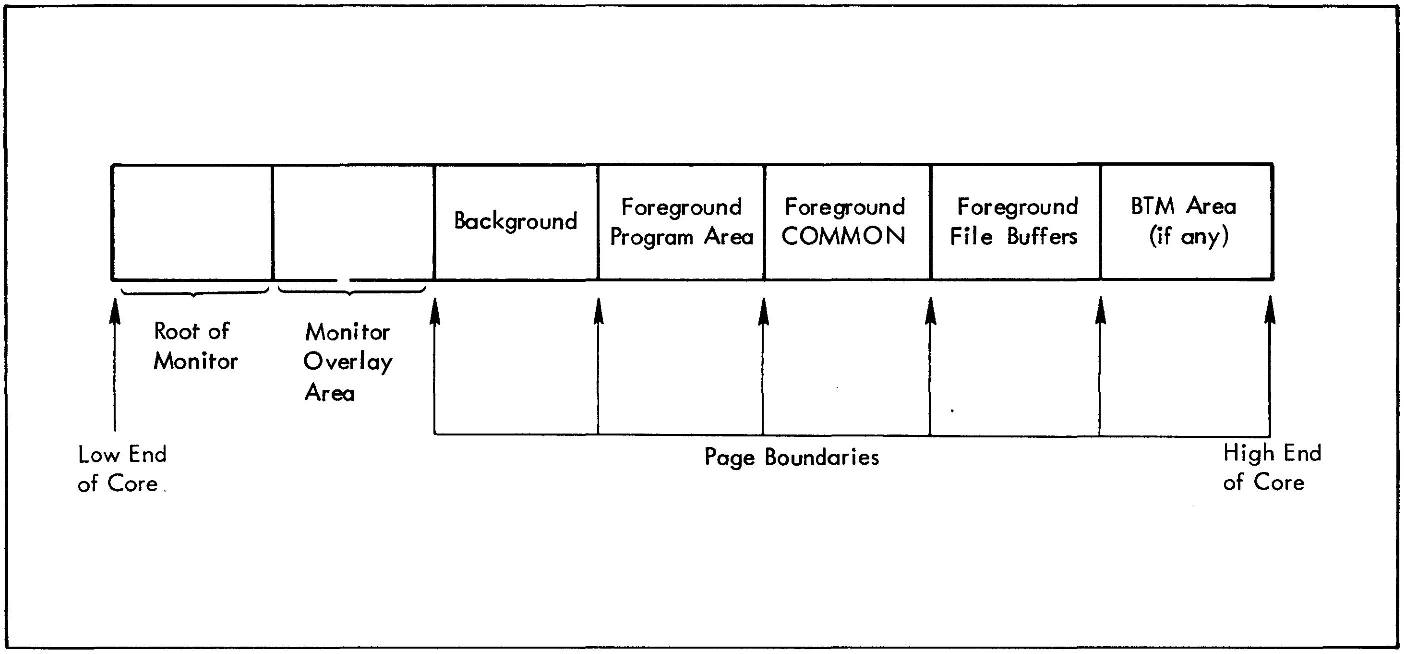


Figure 42. Real-Time System Memory Layout

- There may only be one :SDEVICE command and it must specify that the devices are symbiont and must specify the symbiont to be used in driving the devices. The :SDEVICE is order dependent; the devices must appear in the following order:

  1. Card reader.

  2. Printer.

  3. Card punch.

- In the case of the half-duplex DSC (model 7601), the I/O addresses of the three devices will be the I/O address of the DSC. In the case of the full-duplex DSC (model 7602), the address of the card reader will be the address of the receiver IOP subchannel (even) and of the printer and punch will be the address of the transmitter IOP subchannel (odd).

The following example indicates correct usage of the PASS2 commands:

```
:CHAN

:DEVICE     CRA08     (HAND,DSCIO,DSCCU)

:DEVICE     LPA09

:DEVICE     CPA09
    .
    .
:CHAN

:DEVICE     CRA0A

:DEVICE     LPA0A

:DEVICE     CPA0A
    .
    .
:SDEVICE    (LMN,ISSEG,CRA0A),(LMN,(OSEG,;
:           LPA0A,CPA0A),(LMN,ISSEG,CRA08),;
:           (LMN,OSSEG,LPA09,CPA09)
    .
    .
```

In this example the devices on A08 and A09 are attached to a full-duplex DSC while the devices on A0A are attached to a half-duplex DSC. There is no restriction on the mixture of full- and half-duplex devices and all RBTs will use the same handler routine.

The size (value) for the QUEUE option on the :MONITOR command should be the normal value for a symbiont system plus two queue entries per RBT.

All RB card readers are automatically defined by BPM as M:C devices meaning that all input is in the form of BPM jobs and that any program may access its input data by reading through the C-device operation label. The :STDLB commands used during PASS2 of SYSGEN effect the RB system in the following way:

- Any operational label associated with an output device of the printer or punch type will be associated with the appropriate RBT so that any program producing remote output !JOB commands with the rb-id parameter may use the default operation label assignment.

13. BOOTSTRAP AND PATCHING OPERATIONS

SYSTEM TAPE FORMAT

A master (BI) BPM system tape contains the following elements.

1. A bootstrap loader.
2. Subroutines for patching and initializing the absolute Monitor.
3. A tree table for the absolute Monitor.
4. Root for an absolute Monitor.
5. The Monitor overlay segments.
6. A record identifying the Monitor segment names and numbers for patching purposes.
7. Tape label information.
8. Load modules comprising at least the following labeled files:

    a. M:MON (Monitor, including the PASS0 processor).

    b. LOCCT (loader overlay control command table generator).

    c. PASS1 (processor for file selection and/or updating).

    d. PASS2 (processor for defining the target machine and system parameters).

    e. PASS3 (processor for loading Monitor, processor, and library modules).

    f. DEF (processor for defining the Monitor system to be written to PO).

    g. CCI (control command interpreter).

    h. FMGE (file management processor).

    i. LOADER (object module loader).

    j. ERRMSG (error message file).

    k. PCL (peripheral conversion language).

    l. :DIC (dictionary).

    m. :LIB (library).

    n. All DCBs.

9. Object modules for a System Generation database.

The general arrangement of the information on a master system tape is shown in Figure 31.

SEQUENCE OF OPERATIONS

The Master system tape is loaded into the machine by use of the standard load procedure described in the Reference Manual for the Sigma CPU (see "Loading Operation" in Chapter 5 of the appropriate manual). The hardware bootstrap loads and enters the tape boot at the beginning of the system tape. This tape boot reads in the boot subroutine at TOPRT+200_{16}. Standard I/O error recovery routines are in effect. The boot subroutine then loads the Monitor root and the Monitor tree table at TOPRT (the upper end of root of the Monitor) and enters the Monitor root at INITIAL.

The INITIAL routine initializes part of the Monitor overlay tables and maintains a count of the Monitor overlay segments as they are loaded. Before the first segment is loaded, however, control passes from INITIAL to the boot subroutine at TOPRT + 200_{16}.

The boot subroutine performs a number of functions, as follows. When first entered, the boot subroutine outputs the following message to the operator.

```
C/LL/DC ASSIGN OK (YES/NO)
```

If the operator's response is YES, the subroutine assumes that the device addresses for the control device, listing log, and system RAD or disk pack are not to be changed from those established when the Monitor was defined. If the response is NO, then the following messages will be output.

```
CHANGE ASSIGN PERM (YES/NO)
```

A PO tape (i.e., a generated system tape) may be booted as though it were a BI tape (i.e., a master system tape), and vice versa. The response to this message determines whether the assignments supplied by the operator are to be permanent for the system. If the response is YES, the assignments will be considered permanent; if the response is NO, they will apply only during the bootstrap process. Note that although a permanent reassignment of the system RAD or disk pack is possible (if the response is YES), a temporary RAD or disk pack assignment will be incompatible with the system after booting because the Monitor overlays will not have been loaded to the RAD or disk pack assumed by the operating system.

```
C DEV. =CR
```

In response to this message, the operator must type three characters. These must be the channel and device designation codes (ndd) defining the address of the C device.

```
LL DEV. = LP
```

The operator must make a similar response to define the address of the LL device. The characters typed must be the channel and device designation (ndd).

```
DC DEV. = DC
```

The characters defining the address of the system RAD or disk pack must be the channel and device designation (ndd).

```
DC TYPE =
```

If a RAD is specified, the operator must enter the RAD type (7204, 7212, or 7232).

Before completing any of the above responses with a RET or LF, the operator may cancel the response by striking the RUB key. Following this, or if a completed response is in error, the message

```
??
```

will be output and the key-in request will be repeated.

After all necessary responses have been received, the boot subroutine reads the first patch card via the C device. If there are no patch cards, the first card read must contain an asterisk in column 1.

Cards patching the Monitor root must have the format

```
3F, loc, value
```

where

3F identifies the card as a patch to the root.

loc is the hexadecimal location to be patched.

value is the absolute hexadecimal value to be inserted.

Root patch cards may be present in the patch deck at any point prior to the asterisk card, and may be in any sequence. Cards patching Monitor overlay segments must have the format

```
seg, loc, value
```

where

seg is the hexadecimal segment number of the segment to be patched. The first segment to be read from the system tape is segment 0, the second is segment 1, and so forth.

loc is the hexadecimal location to be patched. This location is expressed relative to the first word of the segment.

value is the absolute hexadecimal value to be inserted.

Segment patch cards must be sequenced in ascending order by segment number, since each segment can be patched only when it is read in from the system tape and held temporarily in core. No provision is made for patching segments after they have been written to the RAD or disk pack.

After all patches have been made to a given segment and it is ready to be written to the RAD or disk pack, the boot subroutine communicates the size and disk address of that segment to the resident root of the absolute Monitor. When all segments have been written to the RAD or disk pack, the asterisk card is read from the C device. This card has the following form:

*[, SAVE][, PSWAP]

where

SAVE specifies that all job files and symbiont files that were previously in the operating system will be saved. The entire P0 (generated system tape) will be read and all files on the P0 tape will replace any files currently existing in the system. New copies of Monitor segments and ABS processors will be written. All PASS2 :DEVICE and :SDEVICE card specifications must be identical (in order, number, and type) in the existing operating system and the new P0 tape. The operating system must be quiescent.

PSWAP specifies that BTM will use the last public disk pack in the system as the swapping device. This option is assumed if there are no RADs in the system. The swapping device initializer (BTMNRES) allocates storage so that a user will not span cylinders and that flawed tracks are bypassed.

Control then passes to another boot subroutine at TOPRT + 202_{16}. This second boot subroutine causes the Monitor root to be copied to the RAD or disk pack, preceded by a disk bootstrap. The subroutine will now read one more record from the input tape and will list the information on the LL device. The information contains the segment name and its corresponding segment number for each segment in the current Monitor tree structure. The next phase of the bootstrap will set up all standard Monitor traps, via XPSD instructions, but no real-time interrupts are connected at this time.

At this point, the resident Monitor is operational but the system environment has not yet been established. To allow the user to define the environment of the operating Monitor, the PASS0 processor (part of Monitor segment CLS) is called and entered at DUMINIT. (The operation of PASS0 is described below.)

When PASSO exits, the message "SIGMA 5/7 BPM" is output and user initialization routines USNRINIT2 and USNRINIT1 are executed (if present). Following this, Monitor routine INITIAL is again entered for the purpose of setting up the Monitor buffer pool area (in the area no longer needed for INITIAL). User initialization routines USRINIT2 and USRINIT1 are executed (if present) and the Monitor then enters the wait state.

BOOTING FROM RAD OR DISK PACK

Once the operating system has been bootstrapped from tape, it may thereafter be copied into core from the RAD or disk pack by means of the load procedure described in Chapter 5 of the CPU Reference Manual.

The hardware boot routine loads and transfers control to the disk boot which then loads the Monitor root into core and enters at INITIAL1. The system is initialized and user routines USRINIT2 and USRINIT1 are executed, but PASSO and user routines USNRINIT2 and USNRINIT1 are not executed following a disk boot. The Monitor then enters the wait state and normal operation may be resumed.

A disk pack (model 7242) can be used. However, to boot the operating system, the computer operator must depress START on the device to put it into the Manual mode and then must depress START again to put it into the Automatic mode.

BOOTSTRAP I/O ERROR RECOVERY

Error recovery during bootstrap procedures is provided for I/O on the card reader, line printer, magnetic tape, disk pack, and RAD. The tape boot reads in only the bootstrap subroutine and passes control to it. While the initial tape boot is in control, limited I/O recovery is available and any error condition detected by TEST I/O (except where the I/O address is not recognized) causes a backspace and an attempt to reread the tape. After 10 retries, the initial tape boot will halt. Once control is passed to the bootstrap subroutine, standard bootstrap I/O error recovery procedures are in effect. The following error messages may appear on the OC device.

```
xx INOPERATIVE
```

```
xx ERROR.  TIO value TDV value
```

```
xx MANUAL MODE
```

```
CHECK-WRITE ERROR
```

where

xx. is MT, CR, DC, LP, or DP.

value indicates the TIO or TDV results.

When either of the first two messages above occur, the wait state is entered. To continue, put the CPU into IDLE, STEP, and then RUN. The I/O will then be retried. If the third message above occurs, I/O will continue when the condition is corrected. When an error occurs for a magnetic tape or disk operation, the operation is retried 10 times before an error message is output. If the fourth message above occurs, the wait state is entered. To continue, put the CPU into IDLE, STEP, and then RUN. The fourth message will appear if the checkwrite on RAD fails. The checkwrite will be executed only if hardware sense switch 1 is set.

NONSTANDARD BOOTSTRAP CONDITIONS

When booting a symbiont system into a machine having a different model RAD than the one for which the system was intended, the bootstrap routines will force the system to be nonsymbiont. This will happen if the operator response is "YES" when responding to the message "CHANGE ASSIGN PERM" (see above).

The message

```
UNKNOWN SEG. IN TAPE BOOTSTRAP
```

will be output if something is wrong with this input tape (not necessarily due to a hardware error). This message should never appear if there are no hardware errors or System Generation errors.

PASSO PROCESSOR

The PASSO processor performs various system initialization functions and constitutes a preliminary part of System Generation, since it defines the environment in which system Generation takes place. However, PASSO is entered automatically whenever a BPM system tape is booted, regardless of whether or not System Generation is to be done.

PASSO reads a user-specified tape (normally the labeled portion of the tape used to bootstrap the absolute Monitor) containing nonresident elements of the BPM system (i.e., CCI, processors, libraries, etc.). It allows the user to modify these elements (and certain system parameters) and writes them to the RAD or disk pack. PASSO then exits to the Monitor. The Monitor then signals the end of PASSO by writing the message

```
* * * * * * * * * * * * * * * * * * 
* *      S I G M A 5/7 BPM       * *
* * * * * * * * * * * * * * * * * *
FIRST AVAILABLE DISC ADDRESS = nnnn
```

(where nnnn indicates the hexadecimal disk address) on the OC device and enters the wait state.

The following commands may be read by PASS0 to define the system environment for the bootstrapped Monitor. Standard device assignments are in effect during PASS0 unless changed by a :GENOP command (see "GENOP" below).

```
:GENCHN        :GENDEF

:GENOP         :GENEXP

:GENDCB        :GENDICT

:GENMD         !END
```

The only required control cards for PASS0 are the :GENDCB and !END cards. Any or all of the others may be omitted. The :GENDCB command is needed to specify the account number, the type of tape device (i.e., 9T or 7T), and the INSN of the tape to be read by PASS0. The !END command terminates PASS0. Any continuation cards should begin with a colon in column 1. Continuation is indicated by means of an appended semicolon.

The only restriction on the order of PASS0 command is that the commands in GROUP1 below must precede the commands in GROUP2.

```
:GENDCB  )
:GENCHN  }  GROUP1
:GENOP   )

:GENMD   )
:GENDICT |
:GENDEF  }  GROUP2
:GENEXP  )

!END
```

PASS0 COMMANDS

The control commands recognized during PASS0 are described below.

:GENCHN This command defines the physical peripheral devices that are to be used by the operating system.

The :GENCHN command has the form

```
:GENCHN yyndd ,...
```

where

yyndd specifies a physical device name (see Tables 14, 15, and 16).

Example:

```
:GENCHN CRA12, DCEA9, 9TG81, 9TAFF
```

:GENOP This command assigns operational labels to peripheral devices. More than one such assignment may be specified in a single :GENOP command.

The :GENOP command has the form

```
:GENOP (label, type) ,...
```

where

label is of the form C, LL, or LO.

type specifies a physical device type (see Table 14).

Default assignments are those defined when the system tape was generated.

Example:

```
:GENOP (C, CRA10), (LO, 9TA80)
```

:GENDCB This command defines the system DCB associated with tape input during PASS0. This command is required each time that a BPM system is booted from tape. The :GENDCB command has the form

```
:, (INSN, value[, value[,...]]),{7T})
                                {9T}

:GENDCB (M:BI, account[, password];
```

where

M:BI specifies that tape input is to be via the M:BI DCB. No other DCB is valid for this command.

account specifies an account identifier (up to 8 alphanumeric characters) associated with the labeled tape to be read during PASS0.

password is the password associated with the labeled tape to be read during PASS0. The password (if any) must correspond to that specified when the tape was created, and may be up to 8 alphanumeric characters in length.

INSN, value, ... specifies the serial number(s) (up to four alphanumeric characters in length) of the tape(s) to be read by PASS0. No more than 3 reels may be specified. The first reel specified must contain the first file to be read, and may be different from the reel used to boot the Monitor. 7T/9T specifies a 7-track or 9-track is to be read.

Example:

```
  :(INSN,001,002),9T)
:GENDCB (M:BI,ACCT1,PASS1,;
```

:GENMD This command inserts program modifications into a specified segment of any load module input from the labeled tape read during PASS0. Since the absolute Monitor is loaded prior to PASS0, it cannot be modified by this command. However, the M:MON load module may be modified by this means.

All such modifications affect any master tapes generated subsequently by the PASS1 processor (see "PASS1" below) and, with the exception of the Monitor itself, affect the current operating system as well.

The :GENMD command has the form

```
  :[,value[{+res(name)}]]
   [      [{+name     }]]
:GENMD,segment loc,value[{+res(name)}];
                        [{+name     }]
```

where

segment specifies the name of the segment that is to be modified.

loc specifies a relative[†] hexadecimal location or a positive absolute hexadecimal address at which the modification is to be made. If it is an absolute address, it must be preceded by a plus (+).

value specifies the word that is to be inserted, right-justified, at the indicated location. If more than one value is given, they will be inserted into successive locations.

res specifies the address resolution for the external definition (see "name" below).[†] If this option is omitted, word resolution is assumed. When the instruction or data word has been relocated, this parameter (res) determines what the resolution of the word is to be.

res	Specified Resolution
BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

name specifies the name of an externally defined symbol whose address or value is to be used to relocate the associated value in the data word.

Examples:

```
:GENMD,SEGA +FA2,+0F0F0F0F
```

```
:GENMD,SEGB LOC1,-3+NAME1
```

```
:GENMD,SEGC LOC2+490,FFFF00+BA(NAME2)
```

```
  :-1+DA(NAME4),+FFFFFFFF+WA(NAME5)
:GENMD,SEGD LOC3-4,10,13+HA(NAME5),;
```

The size of a program that may be modified is limited by the amount of storage available when the PASS0 processor is in control. In order that a program may be modified, both the program and its REF/DEF TABLE must fit in storage between the end of PASS0 and the background upper limit. The PASS0 processor extends beyond the background lower limit by approximately 550 words. In a 32K word system some of the larger programs may not fit and may not be modified with :GENMD. Note that !MODIFY may be used in these cases.

:GENDEF The :GENDEF command may be used to equate an external reference to a specified value.

[†] If an overlay segment is to be modified, the external definition must not have been referenced in a "lower" level of the overlay tree.

The :GENDEF installation control command has the form

```
                                    ⎡⎧+res(name)⎫⎤
:GENDEF, segment xref, value⎢⎨          ⎬⎥
                                    ⎣⎩+name     ⎭⎦
```

where

segment specifies the name of the segment in which the external reference occurs.

xref specifies the name of the external reference that is to be equated to a value.

value specifies the value (relative or absolute hexadecimal) to which the external reference is to be equated.

Examples:

```
:GENDEF, SEGA1 NAMEA1, +3
```

```
:GENDEF, SEGA2 NAMEA2, -FFA
```

```
:GENDEF, SEGA3 NAMEA3, 0+NAMEXX
```

```
:GENDEF, SEGA4 NAMEA4, -2+WA(NAMEZZ)
```

:GENEXP The :GENEXP control command may be used to modify a specified location by the insertion of a new value. It is similar to the :GENMD in that it modifies a core location. In such a case, an external reference to the location just modified may be generated.

The :GENEXP command has the form

```
                                   ⎡⎧+res(name)⎫⎤
:GENEXP, segment loc, value⎢⎨          ⎬⎥
                                   ⎣⎩+name     ⎭⎦
```

where

segment specifies the name of the segment that is to be modified.

loc specifies the relative hexadecimal location or absolute hexadecimal address at which the modification is to be made.

value specifies the word that is to be inserted at the indicated location. If the address field of the inserted word is not currently defined, an expression is generated that has the indicated location as its designation (see "res" and "name" below).

res specifies the address resolution for the external definition (see "name" below). If this option is omitted, word resolution is assumed. When the instruction or data word has been relocated, this parameter (res) determines what the resolution of the word is to be.

res	Specified Resolution
BA	Byte address
HA	Halfword address
WA	Word address
DA	Doubleword address

name specifies the name of an externally defined symbol whose address or value is to be used to relocate its associated value in the data word.

Examples:

```
:GENEXP, SEG1 + 129A, 14
Changes the value in core to 14
```

```
:GENEXP, SEG2 LOCA+3, +0+NAMEA
Changes the value in core to 0 with external reference
to NAMEA, with word resolution.
```

:GENDICT The :GENDICT installation control command may be used to modify a load module's relocation dictionary. It may be used in conjunction with :GENMD and :GENEXP program modifications.

The :GENDICT installation control command has the form

```
:GENDICT, segment loc, code
```

where

segment specifies the name of the segment containing the location referenced (see "loc", below).

loc specifies the relative hexadecimal location or absolute hexadecimal address for which the dictionary entry is to be modified.

code specifies the applicable relocation parameters (see table below).

Code	Relocation	Resolution	Load Module Bias
0	Address	Byte	Module
1	Address	Halfword	Module
2	Address	Word	Module
3	Address	Doubleword	Module
8	Left half	Doubleword	Module
9	Right half	Doubleword	Module
A	Both halves	Doubleword	Module
E	Absolute	-	-

Examples:

```
:GENDICT,SEG1A + 10FF,0
```

```
:GENDICT,SEG1B LOC1A,E
```

```
:GENDICT,SEG1C LOC1B + 59F,9
```

PASS0 MESSAGES

The messages in Table 29 may be output by the PASS0 program, on the LL or OC device. PASS0 either continues its normal operation or initiates an abort return to the Monitor.

Table 29. PASS0 Messages

Message	Description
***ABS DCB/SCRATCH AREA > PSA AREA DEFINED ***SYSGEN PASS-0 ABORTED**	The area where DCB assignment information was to be written or the area to be used for absolute scratch files exceeded the space allocated for permanent RAD of disk pack storage. PASS0 makes an abort return to the Monitor.
----ABS GENERATION COMPLETED	PASS0 has completed its function. PASS0 makes a normal exit to the Monitor.
***CANNOT BOOT LMN	A load module cannot be read from the bootstrap tape because core is not large enough. PASS0 outputs the file name in error and continues to the next file, thus ignoring the file in error.
***CANNOT OPEN**xxx*FILE SKIP TO NEXT ABS PROC	The processor xxx cannot be found in the designated account. PASS0 skips to the next processor (if any) to be absolutized.
***CANNOT READ KEY**zzz*IN**xxx* SKIP TO NEXT ABS PROC	The file named xxx was opened but does not contain the record identified by key zzz. The key protection type 00 and/or 01 for the designated processor cannot be obtained. PASS0 skips to the next processor (if any) to be absolutized.
**GENDCB* CC SYNTAX ERROR	GENDCB contained a syntax error.
**GENDEF* CC SYNTAX ERROR	GENDEF contained a syntax error.
GENDICT CC SYNTAX ERROR	GENDICT contained a syntax error.

Table 29. PASS0 Messages (cont.)

Message	Description
GENEXP CC SYNTAX ERROR	GENEXP contained a syntax error.
GENMD CC SYNTAX ERROR	GENMD contained a syntax error.
INVALID KEYWORD OR VALUE	A keyword or value is unknown. PASS0 ignores the keyword or value and continues.
INVALID YY OR DD FIELD	A device address field was in error syntactically. PASS0 ignores the device address held and continues.
INVALID YYNDD	A physical device address was not defined. PASS0 ignores the device address and continues.
LOC-FIELD SYNTAX ERROR	The command had a location field syntax error. PASS0 ignores the field and continues.
tttttt LOC = nnnnnnnn s lllll	This message indicates which command caused the preceding MODIFICATION LOC OR VALUE INVALID message. tttttt is the command mnemonic. nnnnnnnn is the name associated with the LOC field (if relocatable). s is the sign of the LOC field. lllll is the LOC field value (i.e., the addend (if relocatable) or value (if absolute). PSS0 continues normal processing.
MODIFICATION LOC OR VALUE INVALID	The location and/or value defined in a GENMD, GENDEF, etc. command had a valid syntax but is not valid for the segment specified for modification. The location or value may be out of range, for example, or a name reference may not be found in the segment specified. PASS0 ignores the invalid location or value and continues.
***NO ABS PROCESSORS REQUIRED	This is not an error message. It simply calls attention to the fact that no absolute processors have been requested for the system. PASS0 continues.
NO PAGE AVAILABLE	There was insufficient core space available for use by PASS0. PASS0 aborts and returns to the Monitor.
"OPEN/CLOSE" INFO MISSING	An irrecoverable I/O failure has occurred in opening or closing the BI tape or TM (i.e., temporary) disk files. PASS0 makes an abort return to the Monitor.
PASS-0 CONTROL, NO':'	PASS0 encountered a command without a colon in column 1. PASS0 ignores the command and skips to the next command.
PREVIOUS "MODS" IGNORED	All previously processed GENMD, GENEXP, GENDEF, and GENDICT commands were abrogated. This appears when a syntax error of any type is encountered. PASS0 continues processing.

Table 29. PASS0 Messages (cont.)

Message	Description
***PROC.**xxx* WILL OVERFLOW PSA AREA ***REMAINDER OF ABS PROCESSORS IGNORED	The processor named xxx is too large for the area allocated for permanent RAD or disk pack storage. The designated processor and all remaining processors are ignored. PASS0 continues.
***PROC**xxx* BIAS BELOW BKGRDLL	The processor named xxx is biased below the lower limit of the background area (BKGRDLL). The offending processor is ignored and PASS0 continues.
***PROCESSOR ABSOLUTE	The processor named xxx is absolute. PASS0 continues.
PROCESSOR LMN RELEASED	This message is output following the message "xxx PROCESSOR ABSOLUTE" to indicate that the load module form of processor xxx is released. PASS0 continues.
PROCESSOR LMN SAVED	This message may appear following the message "xxx PROCESSOR ABSOLUTE" to indicate that the load module form of processor xxx was saved (automatic for overlaid processors). PASS0 continues.
"READ" "BI/TM" ABNORMAL	An irrecoverable I/O failure has occurred in reading from the BI tape or the system TM (i.e., temporary) disk files. PASS0 makes an abort return to the Monitor.
SEGMENT-NAME ERROR	PASS0 encountered a file or segment name that is in error. PASS0 ignores the command and continues.
SKIP TO NEXT CC	This message is output in conjunction with other messages.
SYNTAX ERROR	PASS0 encountered a command with a syntax error.
SYSGEN ABORTED (PASS-0)	PASS0 cannot continue its normal processing.
SYSGEN PASS-0 COMPLETED	PASS0 is in the final phase of processing (i.e., the !END command has been received).
SYSGEN PASS-0 IN CONTROL	PASS0 has begun processing.
UNKNOWN CONTROL COMMAND	PASS0 encountered a command that it did not recognize. PASS0 ignores the command and skips to the next command.
VALUE-FIELD SYNTAX ERROR	PASS0 encountered a field with a syntax error. PASS0 ignores the field and continues.
"WRITE" "TM" ABNORMAL	An irrecoverable I/O failure has occurred in writing to the system TM (i.e., temporary) disk files. PASS0 makes an abort return to the Monitor.

14. VOLUME INITIALIZATION

INTRODUCTION

Disk pack devices to be used as private volumes or public devices are initialized by VOLINIT, a special free-standing processor that must run under the !SYS account.

!VOLINIT The !VOLINIT control command calls the Volume Initialization processor, reads its processor commands, and performs the functions specified by the options selected.

The form of the !VOLINIT control command is

```
!VOLINIT
```

Any number of disk pack devices can be initialized under the VOLINIT control command. The processor control commands can be continued using the semicolon (;) character. The form of the processor command is

```
yyndd,[(option)...(option)]
```

where yyndd specifies the name of the device containing the volume to be initialized. If the device is public, the system must be quiescent. Its condition will be checked periodically and if there is a change, an error message will be sent to the operator. If the device is private, it will be marked unavailable until the initialization is complete.

The options are as follows:

$\left\{\begin{array}{l}\text{PUBLIC}\\ \text{PRIVATE}\end{array}\right\}$,sn specifies whether the volume being initialized is public or private. The serial number, sn, parameter is the 1- to 8-byte EBCDIC serial number of the volume. If the volume is private, a Volume Table of Contents (containing the volume serial number and allocation table) is written on granule 0. Empty file directory and account directory table formats will be written on granules 1, 2, and 3 to be used by primary volumes only. (Although a serial number of up to 8 characters is written by VOLINIT to provide for future expansion, at present only the first 4 characters of the serial numbers are used by BPM.)

FLAW,adr-adr,... specifies areas on the device that will be unconditionally flawed (see "Flawing"). Each address, addr, specifies the address of the cylinder/track in the area to be flawed. For example: track 3, cylinder 4 is written 3/4. The first and last track in the area is specified as adr-adr and more than one set of address parameters can be specified. If a single track is desired, the second address parameter can be omitted.

FORMAT,$\left\{\begin{array}{l}\text{adr-adr}\\ \text{NONE}\end{array}\right\}$,... specifies that only the specified area(s) of the device are to be initialized. The parameter NONE specifies that no tracks are to be formatted. If FORMAT is not specified, the entire volume will be initialized.

NOTEST specifies that surface testing will be inhibited. Areas specified by FORMAT are automatically surface tested unless NOTEST is specified. A surface test consists of writing preselected patterns on the device.

ACCT, value specifies the account number to be inserted in the private volume's account directory. The parameter, value, is a 1- to 8-byte EBCDIC account number.

If a processor command contains only the device name, yyndd, and no options are specified, the VOLINIT processor logs the contents of the volume (serial number; date; public/private indicator; home address, if public; account number, if private; and the number of available cylinders, if private) and does not write on the volume.

FLAWING

Cylinders 200-202 are considered a spare pool. When a bad surface is found, one track has flaw marks written in all sector positions and an alternate track address is written in the headers. The alternate track position of the header of the alternate contains the track address of the flawed track. All unused spares contain 1's in the alternate track position.

VOLINIT ERROR MESSAGES

Table 30 contains a list of VOLINIT error messages.

Table 30. VOLINIT Error Messages

Message	Description
!! DEVICE yyndd WRITE PROTECTION VIOLATION	Initialization was suspended due to a write-protect condition on the device. (Currently, no device that can be initialized has write-protect hardware, so this message should never appear.) The operator should contact the Customer Engineer.
!! DEVICE I/O ADDRESS nnn NOT RECOGNIZED	A power failure may have occurred. The operator should restart the job.
!! INVALID I/O INTERRUPT, AIO ADDRESS=nnn	A hardware failure may have occurred. The operator should give output to analyst or Customer Engineer.
!! INVALID KEYIN, PLEASE TRY AGAIN	A previous key-in command was in error. The operator should reissue correct key-in command.
!! JOB ABORTED AT LOCATION nnnnn	A serial hardware or software error has been detected. The operator should give program output to analyst.
!! PLEASE KEY IN CORRECT DATE	The date previously issued was in error. The operator should reenter date.
!! SIO TIME-OUT ON DEVICE nnn	The specified device cannot be started for input our output. The cause may be a hardware failure. The operator should call the Customer Engineer.

APPENDIX A. SIGMA STANDARD OBJECT LANGUAGE

INTRODUCTION

GENERAL

The XDS Sigma standard object language provides a means of expressing the output of any Sigma processor in standard format. All programs and subprograms in this object format can be loaded by the Monitor's relocating loader.[†] Such a loader is capable of providing the program linkages needed to form an executable program in core storage. The object language is designed to be both computer-independent and medium-independent; i.e., it is applicable to any XDS Sigma computer having a 32-bit word length, and the same format is used for both cards and paper tape.

SOURCE CODE TRANSLATION

Before a program can he executed by the computer, it must be translated from symbolic form to binary data words and machine instructions. The primary stages of source program translation are accomplished by a processor. However, under certain circumstances, the processor may not be able to translate the entire source program directly into machine language form.

If a source program contains symbolic forward references, a single-pass processor such as the XDS Symbol assembler can not resolve such references into machine language. This is because the machine language value for the referenced symbol is not established by a one-pass processor until after the statement containing the forward reference has been processed.

A two-pass processor, such as the XDS Meta-Symbol assembler, is capable of making "retroactive" changes in the object program before the object code is output. Therefore, a two-pass processor does not have to output any special object codes for forward references. An example of a forward reference in a Symbol source program is given below.

```
         :
         :
  Y      EQU     $ + 3
         :
         :
         CI, 5   Z
         :
         :
         LI, R   Z
         :
         :
  Z      EQU     2
         :
         :
         BG      Z
         :
         :
  R      EQU     Z + 1
         :
         :
```

In this example the operand $ + 3 is not a forward reference because the assembler can evaluate it when processing the source statement in which it appears. However, the operand Z in the statement

```
    CI,5     Z
```

is a forward reference because it appears before Z has been defined. In processing the statement, the assembler outputs the machine-language code for CI,5, assigns a forward reference number (e.g., 12) to the symbol Z, and outputs that forward reference number. The forward reference number and the symbol Z are also retained in the assembler's symbol table.

When the assembler processes the source statement

```
    LI,R     Z
```

it outputs the machine-language code for LI, assigns a forward reference number (e.g., 18) to the symbol R, outputs that number, and again outputs forward reference number 12 for symbol Z.

On processing the source statement

```
    Z     EQU     2
```

the assembler again outputs symbol Z's forward reference number and also outputs the value, which defines symbol Z, so that the relocating loader will be able to satisfy references to Z in statements CI,5 Z and LI,R Z. At this time, symbol Z's forward reference number (i.e., 12) may be deleted from the assembler's symbol table and the defined value of Z equated with the symbol Z (in the symbol table). Then, subsequent references to Z, as in source statement

```
    BG     Z
```

would not constitute forward references, since the assembler could resolve them immediately by consulting its symbol table.

If a program contains symbolic references to externally defined symbols in one or more separately processed subprograms or library routines, the processor will be unable to generate the necessary program linkages.

An example of an external reference in a Symbol source program is shown below.

```
    REF      ALPH
    :
    :
    LI,3     ALPH
    :
    :
```

When the assembler processes the source statement

```
    REF     ALPH
```

[†] Although a discussion of the object language is not directly pertinent to the BPM, it is included in this manual because it applies to all processors operating under BPM.

it outputs the symbol ALPH, in symbolic (EBCDIC) form, in a declaration specifying that the symbol is an external reference. At this time, the assembler also assigns a declaration name number to the symbol ALPH but does not output the number. The symbol and name number are retained in the assembler's symbol table.

After a symbol has been declared an external reference, it may appear any number of times in the symbolic subprogram in which it was declared. Thus, the use of the symbol ALPH in the source statement

```
    LI,3    ALPH
```

in the above example, is valid even though ALPH is not defined in the subprogram in which it is referenced.

The relocating loader is able to generate interprogram linkages for any symbol that is declared an external definition in the subprogram in which that symbol is defined. Shown below is an example of an external definition in a Symbol source program.

```
              DEF     ALPH
               :
              LI,3    ALPH
               :
    ALPH      AI,4    X'F2'
               :
```

When the assembler processes the source statement

```
    DEF     ALPH
```

it outputs the symbol ALPH, in symbolic (EBCDIC) form, in a declaration specifying that the symbol is an external definition. At this time, the assembler also assigns a declaration name number to the symbol ALPH but does not output the number. The symbol and name number are retained in the assembler's symbol table.

After a symbol has been declared an external definition it may be used (in the subprogram in which it was declared) in the same way as any other symbol. Thus, if ALPH is used as a forward reference, as in the source statement

```
    LI,3      ALPH
```

above, the assembler assigns a forward reference number to ALPH, in addition to the declaration name number assigned previously. (A symbol may be both a forward reference and an external definition.)

On processing the source statement

```
    ALPH      A1,4     X'F2'
```

the assembler outputs the declaration name number of the label ALPH (and an expression for its value) and also outputs the machine-language code for AI,4 and the constant X'F2'.

OBJECT LANGUAGE FORMAT

An object language program generated by a processor is output as a string of bytes representing "load items". A load item consists of an item type code followed by the specific load information pertaining to that item. (The detailed format of each type of load item is given later in this appendix.) The individual load items require varying numbers of bytes for their representation, depending on the type and specific content of each item. A group of 108 bytes, or fewer, comprises a logical record. A load item may be continued from one logical record to the next.

The ordered set of logical records that a processor generates for a program or subprogram is termed an "object module". The end of an object module is indicated by a module-end type code followed by the error severity level assigned to the module by the processor.

RECORD CONTROL INFORMATION

Each record of an object module consists of 4 bytes of control information followed by a maximum of 104 bytes of load information. That is, each record, with the possible exception of the end record, normally consists of 108 bytes of information (i.e., 72 card columns).

The 4 bytes of control information for each record have the form and sequence shown below.

Byte 0

Record Type			Mode		Format		
			1	1	1	0	0
0	1	2	3	4	5	6	7

Byte 1

Sequence Number	
0	7

Byte 2

Checksum	
0	7

Byte 3

Record Size	
0	7

Record Type specifies whether this record is the last record of the module:

000 means last
001 means not last

Mode specifies that the loader is to read binary information. This code is always 11.

Format specifies object language format. This code is always 100.

Sequence Number is 0 for the first record of the module and is incremented by 1 for each record thereafter, until it recycles to 0 after reaching 255.

Checksum is the computed sum of the bytes comprising the record. Carries out of the most significant bit position of the sum are ignored.

Record Size is the number of bytes (including the record control bytes) comprising the logical record (5 ≤ record size ≤ 108). The record size will normally be 108 bytes for all records except the last one, which may be fewer. Any excess bytes in a physical record are ignored.

LOAD ITEMS

Each load item begins with a control byte that indicates the item type. In some instances, certain parameters are also provided in the load item control byte. In the following discussion, load items are categorized according to their function:

1. Declarations identify to the loader the external and control section labels that are to be defined in the object module being loaded.
2. Definitions define the value of forward references, external definitions, the origin of the subprogram being loaded, and the starting address (e.g., as provided in a Symbol/Meta-Symbol END directive).
3. Expression evaluation load items within a definition provide the values (such as constants, forward references, etc.) that are to be combined to form the final value of the definition.
4. Loading items cause specified information to be stored into core memory.
5. Miscellaneous items comprise padding bytes and the module-end indicator.

DECLARATIONS

In order for the loader to provide the linkage between subprograms, the processor must generate for each external reference or definition a load item, referred to as a "declaration", containing the EBCDIC code representation of the symbol and the information that the symbol is either an external reference or a definition (thus, the loader will have access to the actual symbolic name).

Forward references are always internal references within an object module. (External references are never considered forward references.) The processor does not generate a declaration for a forward reference as it does for externals; however, it does assign name numbers to the symbols referenced.

Declaration name numbers (for control sections and external labels) and forward reference name numbers apply only within the object module in which they are assigned. They have no significance in establishing interprogram linkages, since external references and definitions are correlated by matching symbolic names. Hence, name numbers used in any expressions in a given object module always refer to symbols that have been declared within that module.

The processor must generate a declaration for each symbol that identifies a program section. Although the XDS Symbol assembler used with the Monitor allows only a standard control section (i.e., program section), the standard object language includes provision for other types of control sections (such as dummy control sections). Each object module produced by the Symbol processor is considered to consist of at least one control section. If no section is explicitly identified in a Symbol source program, the assembler assumes it to be a standard control section (discussed below). The standard control section is always assigned a declaration name number of 0. All other control sections (i.e., produced by a processor capable of declaring other control sections) are assigned declaration name numbers (1, 2, 3, etc.) in the order of their appearance in the source program.

In the load items discussed below, the access code, pp, designates the memory protection class that is to be associated with the control section. The meaning of this code is given below.

pp	Memory Protection Feature[†]
00	Read, write, or access instructions from.
01	Read or access instructions from.
10	Read only.
11	No access.

Control sections are always allocated on a doubleword boundary. The size specification designates the number of bytes to be allocated for the section.

Declare Standard Control Section

Byte 0

Control byte							
0	0	0	0	1	0	1	1
0	1	2	3	4	5	6	7

Byte 1

Access code				Size (bits 1 through 4)			
p	p	0	0				
0	1	2	3	4	5	6	7

Byte 2

Size (bits 5 through 12)	
0	7

Byte 3

Size (bits 13 through 20)	
0	7

This item declares the standard control section for the object module. There may be no more than one standard control section in each object module. The origin of the standard control section is effectively defined when the first reference to the standard control section occurs, although the declaration item might not occur until much later in the object module.

[†] "Read" means a program can obtain information from the protected area; "write" means a program can store information into a protected area; and, "access" means the computer can execute instructions stored in the protected area.

This capability is required by one-pass processors, since the size of a section cannot be determined until all of the load information for that section has been generated by the processor.

Declare Nonstandard Control Section

Byte 0

Control byte							
0	0	0	0	1	1	0	0
0	1	2	3	4	5	6	7

Byte 1

Access code				Size (bits 1 through 4)	
P	P	0	0		
0	1	2	3	4	7

Byte 2

Size (bits 5 through 12)	
0	7

Byte 3

Size (bits 13 through 20)	
0	7

This item declares a control section other than standard control section (see above). Note that this item is not applicable to the XDS Symbol processor used with the Monitor system. However, the loader is capable of loading object modules (produced by other processors, such as the Meta-Symbol and FORTRAN IV processors) that do contain this item.

Declare Dummy Section

Byte 0

Control byte							
0	0	0	0	1	0	0	1
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

Byte 3

Access code				Size (bits 1 through 4)	
P	P	0	0		
0	1	2	3	4	7

Byte 4

Size (bits 5 through 12)	
0	7

Byte 5

Size (bits 13 through 20)	
0	7

This item comprises a declaration for a dummy control section. It results in the allocation of the specified dummy section, if that section has not been allocated previously by another object module. The label that is to be associated with the first location of the allocated section must be a previously declared external definition name. (Even though the source program may not be required to explicitly designate the label as an external definition, the processor must generate an external definition name declaration for that label prior to generating this load item.)

Declare External Definition Name

Byte 0

Control byte							
0	0	0	0	0	0	1	1
0	1	2	3	4	5	6	7

Byte 1

Name length, in bytes (K)	
0	7

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a label (in EBCDIC code) that is an external definition within the current object module. The name may not exceed 63 bytes in length.

Declare Primary External Reference Name

Byte 0

Control byte							
0	0	0	0	0	1	0	1
0	1	2	3	4	5	6	7

Byte 1

Name length (K), in bytes	
0	7

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a symbol (in EBCDIC code) that is a primary external reference within the current object module. The name may not exceed 63 bytes in length.

A primary external reference is capable of causing the loader to search the system library for a corresponding external definition. If a corresponding external definition is not found in another load module of the program or in the system library, a load error message is output and the job is errored.

Declare Secondary External Reference Name

Byte 0

Control byte							
0	0	0	0	0	1	1	0
0	1	2	3	4	5	6	7

Byte 1

Name length, in bytes (K)	
0	7

Byte 2

First byte of name	
0	7

⋮

Byte K+1

Last byte of name	
0	7

This item declares a symbol (in EBCDIC code) that is a secondary external reference within the current object module. The name may not exceed 63 bytes in length.

A secondary external reference is not capable of causing the loader to search the system library for a corresponding external definition. If a corresponding external definition is not found in another load module of the program, the job is not errored and no error or abnormal message is output.

Secondary external references often appear in library routines that contain optional or alternative subroutines, some of which may not be required by the user's program. By the use of primary external references in the user's program, the user can specify that only those subroutines that are actually required by the current job are to be loaded. Although secondary external references do not cause loading from the library, they do cause linkages to be made between routines that are loaded.

DEFINITIONS

When a source language symbol is to be defined (i.e., equated with a value), the processor provides for such a value by generating an object language expression to be evaluated by the loader. Expressions are of variable length, and terminate with an expression-end control byte (see "Expression Evaluation" in this appendix). An expression is evaluated by the addition or subtraction of values specified by the expression.

Since the loader must derive values for the origin and starting address of a program, these also require definition.

Origin

Byte 0

Control byte							
0	0	0	0	0	1	0	0
0	1	2	3	4	5	6	7

This item sets the loader's load-location counter to the value designated by the expression immediately following the origin control byte. This expression must not contain any elements that cannot be evaluated by the loader (see "Expression Evaluation" which follows).

Forward Reference Definition

Byte 0

Control byte							
0	0	0	0	1	0	0	0
0	1	2	3	4	5	6	7

Byte 1

First byte of reference number	
0	7

Byte 2

Second byte of reference number	
0	7

This item defines the value (expression) for a forward reference. The referenced expression is the one immediately following byte 2 of this load item, and must not contain any elements that cannot be evaluated by the loader (see "Expression Evaluation" which follows).

Forward Reference Definition and Hold

Byte 0

Control byte							
0	0	0	1	0	0	0	0
0	1	2	3	4	5	6	7

Byte 1

First byte of reference number	
0	7

Byte 2

Second byte of reference number	
0	7

This item defines the value (expression) for a forward reference and notifies the loader that this value is to be retained in the loader's symbol table until the module end is encountered. The referenced expression is the one immediately following the name number. It may contain values that have not been defined previously, but all such values must be available to the loader prior to the module end.

After generating this load item, the processor need not retain the value for the forward reference, since that responsibility is then assumed by the loader. However, the processor must retain the symbolic name and forward reference number assigned to the forward reference (until module end).

External Definition

Byte 0

Control byte							
0	0	0	0	1	0	1	0
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

This item defines the value (expression) for an external definition name. The name number refers to a previously declared definition name. The referenced expression is the one immediately following the name number.

Define Start

Byte 0

Control byte							
0	0	0	0	1	1	0	1
0	1	2	3	4	5	6	7

This item defines the starting address (expression) to be used at the completion of loading. The referenced expression is the one immediately following the control byte.

EXPRESSION EVALUATION

A processor must generate an object language expression whenever it needs to communicate to the loader one of the following:

1. A program load origin.
2. A program starting address.
3. An external definition value.
4. A forward reference value.
5. A field definition value.

Such expressions may include sums and differences of constants, addresses, and external or forward reference values that, when defined, will themselves be constants or addresses.

After initiation of the expression mode, by the use of a control byte designating one of the five items described above, the value of an expression is expressed as follows:

1. An address value is represented by an offset from the control section base plus the value of the control section base.

2. The value of a constant is added to the accumulated sum by generating an Add Constant (see below) control byte followed by the value, right-justified in four bytes.

   The offset from the control section base is given as a constant representing the number of units of displacement from the control section base, at the resolution of the address of the item. That is, a word address would have its constant portion expressed as a count of the number of words offset from the base, while the constant portion of a byte address would be expressed as the number of bytes offset from the base.

   The control section base value is accumulated by means of an Add Value of Declaration (see below) or Subtract Value of Declaration load item specifying the desired resolution and the declaration number of the control section base. The loader adjusts the base value to the specified address resolution before adding it to the current partial sum for the expression.

   In the case of an absolute address, an Add Absolute Section (see below) or Subtract Absolute Section control byte must be included in the expression to identify the value as an address and to specify its resolution.

3. An external definition or forward reference value is included in an expression by means of a load item adding or subtracting the appropriate declaration or forward reference value. If the value is an address, the resolution specified in the control byte is used to align the value before adding it to the current partial sum for the expression. If the value is a constant, no alignment is necessary.

Expressions are not evaluated by the loader until all required values are available. In evaluating an expression, the loader maintains a count of the number of values added or subtracted at each of the four possible resolutions. A separate counter is used for each resolution, and each counter is incremented or decremented by 1 whenever a value of the corresponding resolution is added to or subtracted from the loader's expression accumulator. The final accumulated sum is a constant, rather than an address value, if the final count in all four counters is equal to 0. If the final count in one (and only one) of the four counters is equal to +1 or -1, the

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

accumulated sum is a "simple address" having the resolution of the nonzero counter. If more than one of the four counters have a nonzero final count, the accumulated sum is termed a "mixed-resolution expression" and is treated as a constant rather than an address.

The resolution of a simple address may be altered by means of a Change Expression Resolution (see below) control byte. However, if the current partial sum is either a constant or a mixed-resolution value when the Change Expression Resolution control byte occurs, then the expression resolution is unaffected.

Note that the expression for a program load origin or starting address must resolve to a simple address, and the single nonzero resolution counter must have a final count of +1 when such expressions are evaluated.

In converting a byte address to a word address, the two least significant bits of the address are truncated. Thus, if the resulting word address is later changed back to byte resolution, the referenced byte location will then be the first byte (byte 0) of the word.

After an expression has been evaluated, its final value is associated with the appropriate load item.

In the following diagrams of load item formats, RR refers to the address resolution code. The meaning of this code is given in the table below.

RR	Address Resolution
00	Byte
01	Halfword
10	Word
11	Doubleword

The load items discussed in this appendix, "Expression Evaluation", may appear only in expressions.

Add Constant

Byte 0

Control byte							
0	0	0	0	0	0	0	1
0	1	2	3	4	5	6	7

Byte 1

First byte of constant	
0	7

Byte 2

Second byte of constant	
0	7

Byte 3

Third byte of constant	
0	7

Byte 4

Fourth byte of constant	
0	7

This item causes the specified 4-byte constant to be added to the loader's expression accumulator. Negative constants are represented in two's complement form.

Add Absolute Section

Byte 0

Control byte							
0	0	1	1	0	1	R	R
0	1	2	3	4	5	6	7

This item identifies the associated value (expression) as a positive absolute address. The address resolution code, RR, designates the desired resolution.

Subtract Absolute Section

Byte 0

Control byte							
0	0	1	1	1	0	R	R
0	1	2	3	4	5	6	7

This item identifies the associated value (expression) as a negative absolute address. The address resolution code, RR, designates the desired resolution.

Add Value of Declaration

Byte 0

Control byte							
0	0	1	0	0	0	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

This item causes the value of the specified declaration to be added to the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the name number refers to a previously declared definition name that is to be associated with the first location of the allocated section.

One such item must appear in each expression for a relocatable address occurring within a control section, adding the value of the specified control section declaration (i.e., adding the byte address of the first location of the control section).

Add Value of Forward Reference

Byte 0

Control byte							
0	0	1	0	0	1	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of forward reference number	
0	7

Byte 2

Second byte of forward reference number	
0	7

This item causes the value of the specified forward reference to be added to the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the designated forward reference must not have been defined previously.

Subtract Value of Declaration

Byte 0

Control byte							
0	0	1	0	1	0	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]

This item causes the value of the specified declaration to be subtracted from the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the name number refers to a previously declared definition name that is to be associated with the first location of the allocated section.

Subtract Value of Forward Reference

Byte 0

Control byte							
0	0	1	0	1	1	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of forward reference number	
0	7

Byte 2

Second byte of forward reference number	
0	7

This item causes the value of the specified forward reference to be subtracted from the loader's expression accumulator. The address resolution code, RR, designates the desired resolution, and the designated forward reference must not have been defined previously.

Change Expression Resolution

Byte 0

Control byte							
0	0	1	1	0	0	R	R
0	1	2	3	4	5	6	7

This item causes the address resolution in the expression to be changed to that designated by RR.

Expression End

Byte 0

Control byte							
0	0	0	0	0	0	1	0
0	1	2	3	4	5	6	7

This item identifies the end of an expression (the value of which is contained in the loader's expression accumulator).

FORMATION OF INTERNAL SYMBOL TABLES

The three object code control bytes described below are required to supply the information necessary in the formation of Internal Symbol Tables.

In the following diagrams of load item formats, Type refers to the symbol types supplied by the object language and maintained in the symbol table. IR refers to the internal resolution code. Type and resolution are meaningful only when the value of a symbol is an address. In this case, it is highly likely that the processor knows the type of value that is in the associated memory location, and the type field identifies it. The resolution field indicates the resolution of the location counter at the time the symbol was defined. The following tables summarize the combinations of value and meaning.

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

Symbol Types

Type	Meaning of 5-Bit Code
00000	Instruction
00001	Integer
00010	Short floating point
00011	Long floating point
00110	Hexadecimal (also for packed decimal)
00111	EBCDIC text (also for unpacked decimal)
01001	Integer array
01010	Short floating-point array
01011	Long floating-complex array
01000	Logical array
10000	Undefined symbol

Internal Resolution

IR	Address Resolution
000	Byte
001	Halfword
010	Word
011	Doubleword
100	Constant

Type Information for External Symbol

Byte 0

Control byte							
0	0	0	1	0	0	0	1

0 1 2 3 4 5 6 7

Byte 1

Type field	IR field

0 4 5 7

Byte 2

Name number

0 7

Byte 3 (if required)

Name number (continued)

0 7

This item provides type information for external symbols. The Type and IR fields are defined above. The name number field consists of one or two bytes (depending on the current declaration count) which specifies the declaration number of the external definition.

Type and EBCDIC for Internal Symbol

Byte 0

Control byte							
0	0	0	1	0	0	1	0

0 1 2 3 4 5 6 7

Byte 1

Type field	IR field

0 4 5 7

Byte 2

Length of name (EBCDIC characters)

0 7

Byte 3

First byte of name in EBCDIC

0 7

Byte n

Last byte of name in EBCDIC

0 7

Byte n + 1, . . .

Expression defining value of internal symbol

0 7

This item supplies type and EBCDIC,for an internal symbol. The load items for Type and IR are as above. Length of name specifies the length of the EBCDIC name in characters. The name, in EBCDIC, is specified in the required number of bytes, followed by the expression defining the internal symbol.

EBCDIC for an Undefined Symbol

Byte 0

Control byte							
0	0	0	1	0	0	1	1

0 1 2 3 4 5 6 7

Byte 1

Length of name (EBCDIC characters)

0 7

Byte 2

First byte of name in EBCDIC

0 7

Byte n

Last byte of name in EBCDIC

0 7

Byte n + 1, n + 2

Two bytes of symbol associated forward reference number

0 7

This item is used to associate a symbol with a forward reference. The length of name and name in EBCDIC are the same as in the above item. The last two bytes specify the forward reference number with which the above symbol is to be associated.

LOADING

Load Absolute

Byte 0

Control byte							
0	1	0	0	N	N	N	N
0	1	2	3	4	5	6	7

Byte 1

First byte to be loaded	
0	7

⋮

Byte NNNN

Last byte to be loaded	
0	7

This item causes the next NNNN bytes to be loaded absolutely (NNNN is expressed in natural binary form, except that 0000 is interpreted as 16 rather than 0). The load location counter is advanced appropriately.

Load Relocatable (Long Form)

Byte 0

Control byte							
0	1	0	1	Q	C	R	R
0	1	2	3	4	5	6	7

Byte 1

First byte of name number	
0	7

Byte 2

Second byte of name number[†]	
0	7

This item causes a 4-byte word (immediately following this load item) to be loaded, and relocates the address field according to the address resolution code, RR. Control bit C designates whether relocation is to be relative to a forward reference (C = 1) or relative to a declaration (C = 0). Control bit Q designates whether a 1-byte (Q = 1) or a 2-byte (Q = 0) name number follows the control byte of this load item.

If relocation is to be relative to a forward reference, the forward reference must not have been defined previously. When this load item is encountered by the loader, the load location counter can be aligned with a word boundary by loading the appropriate number of bytes containing all zeros (e.g., by means of a load absolute item).

Load Relocatable (Short Form)

Byte 0

Control byte							
1	C	D	D	D	D	D	D
0	1	2	3	4	5	6	7

This item causes a 4-byte word (immediately following this load item) to be loaded, and relocates the address field (word resolution). Control bit C designates whether relocation is to be relative to a forward reference (C = 1) or relative to a declaration (C = 0). The binary number DDDDDD is the forward reference number or declaration number by which relocation is to be accomplished.

If relocation is to be relative to a forward reference, the forward reference must not have been defined previously. When this load item is encountered by the loader, the load location counter must be on a word boundary (see "Load Relocatable (Long Form)", above).

Repeat Load

Byte 0

Control byte							
0	0	0	0	1	1	1	1
0	1	2	3	4	5	6	7

Byte 1

First byte of repeat count	
0	7

Byte 2

Second byte of repeat count	
0	7

This item causes the loader to repeat (i.e., perform) the subsequent load item a specified number of times. The repeat count must be greater than 0, and the load item to be repeated must follow the repeat load item immediately.

Define Field

Byte 0

Control byte							
0	0	0	0	0	1	1	1
0	1	2	3	4	5	6	7

Byte 1

Field location constant, in bits (K)	
0	7

Byte 2

Field length, in bits (L)	
0	7

This item defines a value (expression) to be added to a field in previously loaded information. The field is of length L ($1 \leq L \leq 255$) and terminates in bit position T, where:

T = current load bit position -256 +K.

[†] If the module has fewer than 256 previously assigned name numbers, this byte is absent.

The field location constant, K, may have any value from 1 to 255. The expression to be added to the specified field is the one immediately following byte 2 of this load item.

MISCELLANEOUS LOAD ITEMS

Padding

Byte 0

Control byte							
0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7

Padding bytes are ignored by the loader. The object language allows padding as a convenience for processors.

Module End

Byte 0

Control byte							
0	0	0	0	1	1	1	0
0	1	2	3	4	5	6	7

Byte 1

Severity level							
0	0	0	0	E	E	E	E
0	1	2	3	4	5	6	7

This item identifies the end of the object module. The value EEEE is the error severity level assigned to the module by the processor.

OBJECT MODULE EXAMPLE

The following example shows the correspondence between the statements of a Symbol source program and the string of object bytes output for that program by the assembler. The program, listed below, has no significance other than illustrating typical object code sequences.

Example

```
 1                                   DEF     AA,BB,CC   CC IS UNDEFINED BUT CAUSES NO
                                                        ERROR
 2                                   REF     RZ,RTN     EXTERNAL REFERENCES DECLARED
 3  00000                     ALPHA  CSECT              DEFINE CONTROL SECTION ALPHA
 4  000C8                            ORG     200        DEFINE ORGIN
 5  000C8     22000000  N     AA     LI,CNT  0          DEFINES EXTERNAL AA; CNT IS A
                                                        FWD REF
 6  000C9     32000000  N            LW,R    RZ         R IS A FORWARD REFERENCE;
 7                            *                         RZ IS AN EXTERNAL REFERENCE, AS
 8                            *                         DECLARED IN LINE 2
 9  000CA     50000000  N     RPT    AH,R    KON        DEFINES RPT; R AND KON ARE
10                            *                         FORWARD REFERENCES
11  000CB     69200000  F            BCS,2   BB         BB IS AN EXTERNAL DEFINITION
12                            *                         USED AS A FORWARD REFERENCE
13  000CC     20000001  N            AI,CNT  1          CNT IS A FORWARD REFERENCE
14  000CD     680000CA               B       RPT        RPT IS A BACKWARD REFERENCE
15  000CE     68000000  X            B       RTN        RTN IS AN EXTERNAL REFERENCE
16  000CF     0001      A     KON    DATA,2  1          DEFINES KON
17          00000003          R      EQU     3          DEFINES R
18          00000004          CNT    EQU     4          DEFINES CNT
19  000D0     224FFFFF  A     BB     LI,CNT  -1         DEFINES EXTERNAL BB THAT HAS
20                            *                         ALSO BEEN USED AS A FORWARD
21                            *                         REFERENCE
22  000C8                            END     AA         END OF PROGRAM
```

CONTROL BYTES (In Binary)

```
Begin Record   Record number: 0

00111100   Record type: not last, Mode binary, Format: object language.        Record control
00000000   Sequence number 0                                                   information not
01100011   Checksum: 99                                                        part of load item
01101100   Record size: 108

           03020101 (hexadecimal code comprising the load item)
00000011   Declare external definition name (2 bytes)  Name: AA    Declaration number: 1

           03020202
00000011   Declare external definition name (2 bytes)  Name: BB    Declaration number: 2   Source Line 1

           03020303
00000011   Declare external definition name (2 bytes)  Name: CC    Declaration number: 3

           0502D9E9
00000101   Declare primary reference name (2 bytes)  Name RZ       Declaration number: 4
                                                                                       Source Line 2
           0503D9E3D5
00000101   Declare primary reference name (3 butes)  Name: RTN     Declaration number: 5

           0A010100000320200002
00001010   Define external definition
           Number 1
00000001   Add constant: 800   X'320'                                                  Source Line 5†
00100000   Add value of declaration (byte resolution)
           Number 0
00000010   Expression end

           040100000320200002
00000100   Origin
00000001   Add constant: 800   X'320'
00100000   Add value of declaration (byte resolution)                                  Source Line 4
           Number 0
00000010   Expression end

           4422000000
01000100   Load absolute the following 4 bytes: X'22000000'

           07EB0426000002
00000111   Define field
           Field location constant: 235   bits
           Field length: 4   bits                                                      Source Line 5
           Add the following expression to the above field:
00100110   Add value of forward reference (word resolution)
           Number 0
00000010   Expression end
```

†No object code is generated for source lines 3 (define control section) or 4 (define origin) at the time they are encountered. The control section is declared at the end of the program after Symbol has determined the number of bytes the program requires. The origin definition is generated prior to the first instruction.

```
            8432000000
10000100    Load relocatable (short form).  Relocate address field (word resolution)
            Relative to declaration number 4
            The following 4 bytes: X'32000000'

            07EB0426000602
00000111    Define field                                                                Source Line 6
            Field location constant: 235   bits
            Field length: 4   bits
            Add the following expression to the above field:
00100110    Add value of forward reference (word resolution)
            Number 6
00000010    Expression end

            CC50000000
11001100    Load relocatable (short form).  Relocate address field (word resolution)
            Relative to forward reference number 12
            The following 4 bytes: X'50000000'

            07EB0426000602
00000111    Define field                                                                Source Line 9
            Field location constant: 235   bits
            Field length: 4   bits
            Add the following expression to the above field:
00100110    Add value of forward reference (word resolution)
            Number 6
00000010    Expression end

            D269200000
11010010    Load relocatable (short form).  Relocate address field (word resolution)
            Relative to forward reference number 18                                     Source Line 11
            The following 4 bytes: X'69200000'

            4420000001
01000100    Load absolute the following 4 bytes: X'20000001'

            07EB0426000002
00000111    Define field
            Field location constant: 235   bits                                         Source Line 13
            Field length: 4   bits
            Add the following expression to the above field:
00100110    Add value of forward reference (word resolution)
            Number 0
00000010    Expression end

            80680000CA
10000000    Load relocatable (short form).  Relocate address field (word resolution)
            Relative to declaration number 0                                            Source Line 14
            The following 4 bytes: X'680000CA'

            8568000000
10000101    Load relocatable (short form).  Relocate address field (word resolution)    Source Line 15
            Relative to declaration number 5
            The following 4 bytes: X'68000000'

            08
00001000    Define forward reference (continued in record 1)                            Source Line 16
```

```
Begin Record   Record number 1

00011100       Record type: last,  Mode: binary,  Format: object language.
00000001       Sequence number 1                                              Record Control
11101100       Checksum: 236                                                  Information
01010001       Record size: 81

               000C010000033C200002 (continued from record 0)
               Number 12
00000001       Add constant: 828   X'33C'
00100000       Add value of declaration (byte resolution)                     Source Line 16
               Number 0
00000010       Expression end

               42001
01000010       Load absolute the following 2 bytes: X'0001'

               080006010000000302
00001000       Define forward reference
               Number 6                                                       Source Line 17
00000001       Add constant: 3   X'3'
00000010       Expression end

               080000010000000402
00001000       Define forward reference
               Number 0                                                       Source Line 18
00000001       Add constant: 4   X'4'
00000010       Expression end

               0F00024100
00001111       Repeat load                                                    Advance to Word
               Repeat count: 2                                                Boundary
01000001       Load absolute the following 1 bytes: X'00'

               0800120100000340200002
00001000       Define forward reference
               Number 18
00000001       Add constant: 832   X'340'
               Add value of declaration (byte resolution)
               Number 0
00000010       Expression end

               0A020100000340200002
00001010       Define external definition                                     Source Line 19
               Number 2
00000001       Add constant: 832   X'340'
00100000       Add value of declaration (byte resolution)
               Number 0
00000010       Expression end

               44224FFFFF
01000100       Load absolute the following 4 bytes: X'224FFFFF'

               0D0100000320200002
00001101       Define start
00000001       Add constant: 800   X'320'
00100000       Add value of declaration (byte resolution)                     Source Line 22
               Number 0
00000010       Expression end
```

```
            0B000344
00001011    Declare standard control section declaration number: 0
            Access code:  Full access.   Size 836   X'344'

            0E00
00001110    Module end
                Severity level:  X'0'
```

A table summarizing control byte codes for object language load items is given below.

Object Code Control Byte								Type of Load Item
0	0	0	0	0	0	0	0	Padding
0	0	0	0	0	0	0	1	Add constant
0	0	0	0	0	0	1	0	Expression end
0	0	0	0	0	0	1	1	Declare external definition name
0	0	0	0	0	1	0	0	Origin
0	0	0	0	0	1	0	1	Declare primary reference name
0	0	0	0	0	1	1	0	Declare secondary reference name
0	0	0	0	0	1	1	1	Define field
0	0	0	0	1	0	0	0	Define forward reference
0	0	0	0	1	0	0	1	Declare dummy section
0	0	0	0	1	0	1	0	Define external definition
0	0	0	0	1	0	1	1	Declare standard control section
0	0	0	0	1	1	0	0	Declare nonstandard control section
0	0	0	0	1	1	0	1	Define start
0	0	0	0	1	1	1	0	Module end
0	0	0	0	1	1	1	1	Repeat load
0	0	0	1	0	0	0	0	Define forward reference and hold
0	0	0	1	0	0	0	1	Provide type information for external symbol
0	0	0	1	0	0	1	0	Provide type and EBCDIC for internal symbol
0	0	0	1	0	0	1	1	EBCDIC and forward reference number for undefined symbol
0	0	1	0	0	0	R	R	Add value of declaration
0	0	1	0	0	1	R	R	Add value of forward reference
0	0	1	0	1	0	R	R	Subtract value of declaration
0	0	1	0	1	1	R	R	Subtract value of forward reference
0	0	1	1	0	0	R	R	Change expression resolution
0	0	1	1	0	1	R	R	Add absolute section
0	0	1	1	1	0	R	R	Subtract absolute section
0	1	0	0	N	N	N	N	Load absolute
0	1	0	1	Q	C	R	R	Load relocatable (long form)
1	C	D	D	D	D	D	D	Load relocatable (short form)

APPENDIX B. XDS SIGMA STANDARD COMPRESSED LANGUAGE

The XDS Sigma Standard Compressed Language is used to represent source EBCDIC information in a highly compressed form.

Meta-Symbol (along with several of the utility programs) accepts this form as input or output, will accept updates to the compressed input and will regenerate source when requested. No information is destroyed in the compression or decompression.

Records may not exceed 108 bytes in length. Compressed records are punched in the binary mode when represented on card media. Therefore, on cards, columns 73 through 80 are not used and are available for comment or identification information.

The first four bytes of each record are for checking purposes. They are as follows:

Byte 1 Identification (00L11000) L = 1 for each record except the last record, in which case L = 0.

Byte 2 Sequence number (0 to 255 and recycles).

Byte 3 Checksum which is the least significant 8 bits of the sum of all bytes in the record except the checksum byte itself. Carries out of the most significant bit are ignored. If the checksum byte is all 1's, do not checksum the record.

Byte 4 Number of bytes comprising record including the checking bytes (≤ 108)

The rest of the record consists of a string of 6-bit and 8-bit items. Any partial item at the end of a record is ignored.

The following 6-bit items (decimal number assigned) comprise the string control:

Item	Function	Item	Function
0	Ignore	32	O
1	Not currently assigned	33	P
2	End of line	34	Q
3	End of file	35	R
4	Use 8-bit character that follows	36	S
5	Use n + 1 blanks (next 6-bit item is n)	37	T
6	Use n + 65 blanks (next 6-bit item is n)	38	U
7	Blank	39	V
8	0	40	W
9	1	41	X
10	2	42	Y
11	3	43	Z
12	4	44	.
13	5	45	<
14	6	46	(
15	7	47	+
16	8	48	|
17	9	49	&
18	A	50	$
19	B	51	*
20	C	52	)
21	D	53	;
22	E	54	¬
23	F	55	-
24	G	56	/
25	H	57	,
26	I	58	%
27	J	59	⌴
28	K	60	>
29	L	61	:
30	M	62	'
31	N	63	=

APPENDIX C. REFERENCE TABLES

This appendix contains the following reference material:

XDS STANDARD SYMBOLS AND CODES

The symbol and code standards described in this publication are applicable to all XDS products, both hardware and software. They may be expanded or altered from time to time to meet changing requirements.

The symbols listed here include two types: graphic symbols and control characters. Graphic symbols are displayable and printable; control characters are not. Hybrids are SP, the symbol for a blank space; and DEL, the delete code, which is not considered a control command.

Three types of code are shown: (1) the 8-bit XDS Standard Computer Code, i.e., the XDS Extended Binary-Coded-Decimal Interchange Code (EBCDIC); (2) the 7-bit American National Standard Code for Information Interchange (ANSCII); and (3) the XDS standard card code.

XDS STANDARD CHARACTER SETS

1. EBCDIC

   57-character set: uppercase letters, numerals, space, and & - / . < > ( ) + | $ * : ; , % # @ ' =

   63-character set: same as above plus ¢ ! _ ? " ¬

   89-character set: same as 63-character set plus lowercase letters

2. ANSCII

   64-character set: uppercase letters, numerals, space, and ! " $ % & ' ( ) * + , - . / \ ; : = < > ? @ _ [ ] ^ #

   95-character set: same as above plus lowercase letters and { } ¦ ~ `

CONTROL CODES

In addition to the standard character sets listed above, the XDS symbol repertoire includes 37 control codes and the hybrid code DEL (hybrid code SP is considered part of all character sets). These are listed in the table titled XDS Standard Symbol-Code Correspondences.

SPECIAL CODE PROPERTIES

The following two properties of all XDS standard codes will be retained for future standard code extensions:

1. All control codes, and only the control codes, have their two high-order bits equal to "00". DEL is not considered a control code.

2. No two graphic EBCDIC codes have their seven low-order bits equal.

XDS STANDARD 8-BIT COMPUTER CODES (EBCDIC)

			Most Significant Digits															
	Hexadecimal		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Least Significant Digits	0	0000	NUL	DLE	ds		SP	&	-	▨								0
	1	0001	SOH	DC1	ss		▨	▨	/	▨	a	j		\[1]	A	J		1
	2	0010	STX	DC2	fs		▨	▨	▨	▨	b	k	s	{[1]	B	K	S	2
	3	0011	ETX	DC3	si		▨	▨	▨	▨	c	l	t	}[1]	C	L	T	3
	4	0100	EOT	DC4			▨	▨	▨	▨	d	m	u	[[1]	D	M	U	4
	5	0101	HT	LF NL			Will not be assigned				e	n	v	][1]	E	N	V	5
	6	0110	ACK	SYN			▨	▨	▨	▨	f	o	w		F	O	W	6
	7	0111	BEL	ETB			▨	▨	▨	▨	g	p	x		G	P	X	7
	8	1000	EOM BS	CAN			▨	▨	▨	▨	h	q	y		H	Q	Y	8
	9	1001	ENQ	EM			▨	▨	▨	▨	i	r	z		I	R	Z	9
	A	1010	NAK	SUB			¢[2]	!	^[1]	:					▨	▨	▨	▨
	B	1011	VT	ESC			.	$	,	#					▨	▨	▨	▨
	C	1100	FF	FS			<	*	%	@					Will not be assigned			
	D	1101	CR	GS			(	)	_	'					▨	▨	▨	▨
	E	1110	SO	RS			+	;	>	=					▨	▨	▨	▨
	F	1111	SI	US			|[2]	¬[2]	?	"					▨	▨	▨	DEL
			3				4				5							

NOTES:

1 The characters ^ \ { } [ ] are ANSCII characters that do not appear in any of the XDS EBCDIC-based character sets, though they are shown in the EBCDIC table.

2 The characters ¢ | ¬ appear in the XDS 63- and 89-character EBCDIC sets but not in either of the XDS ANSCII-based sets. However, XDS software translates the characters ¢ | ¬ into ANSCII characters as follows:

EBCDIC	=	ANSCII
¢		` (6-0)
|		¦ (7-12)
¬		~ (7-14)

3 The EBCDIC control codes in columns 0 and 1 and their binary representation are exactly the same as those in the ANSCII table, except for two interchanges: LF/NL with NAK, and HT with ENQ.

4 Characters enclosed in heavy lines are included only in the XDS standard 63- and 89-character EBCDIC sets.

5 These characters are included only in the XDS standard 89-character EBCDIC set.

XDS STANDARD 7-BIT COMMUNICATION CODES (ANSCII) [1]

			Most Significant Digits							
	Decimal (rows) ↓	(col's.) →	0	1	2	3	4	5	6	7
		Binary [1]	x000	x001	x010	x011	x100	x101	x110	x111
Least Significant Digits	0	0000	NUL	DLE	SP	0	@	P	`	p
	1	0001	SOH	DC1	![5]	1	A	Q	a	q
	2	0010	STX	DC2	"	2	B	R	b	r
	3	0011	ETX	DC3	#	3	C	S	c	s
	4	0100	EOT	DC4	$	4	D	T	d	t
	5	0101	ENQ	NAK	%	5	E	U	e	u
	6	0110	ACK	SYN	&	6	F	V	f	v
	7	0111	BEL	ETB	'	7	G	W	g	w
	8	1000	BS	CAN	(	8	H	X	h	x
	9	1001	HT	EM	)	9	I	Y	i	y
	10	1010	LF NL	SUB	*	:	J	Z	j	z
	11	1011	VT	ESC	+	;	K	[[5]	k	{
	12	1100	FF	FS	,	<	L	\	l	¦
	13	1101	CR	GS	-	=	M	][5]	m	}
	14	1110	SO	RS	.	>	N	[4]^[5]	n	~[4]
	15	1111	SI	US	/	?	O	_[4]	o	DEL
			2		3					

NOTES:

1 Most significant bit, added for 8-bit format, is either 0 or even parity.

2 Columns 0-1 are control codes.

3 Columns 2-5 correspond to the XDS 64-character ANSCII set.
Columns 2-7 correspond to the XDS 95-character ANSCII set.

4 On many current teletypes, the symbol

^	is	↑	(5-14)
_	is	←	(5-15)
~	is	ESC or ALTMODE control	(7-14)

and none of the symbols appearing in columns 6-7 are provided. Except for the three symbol differences noted above, therefore, such teletypes provide all the characters in the XDS 64-character ANSCII set. (The XDS 7015 Remote Keyboard Printer provides the 64-character ANSCII set also, but prints ^ as ∧.)

5 On the XDS 7670 Remote Batch Terminal, the symbol

!	is	|	(2-1)
[	is	¢	(5-11)
]	is	!	(5-13)
^	is	¬	(5-14)

and none of the symbols appearing in columns 6-7 are provided. Except for the four symbol differences noted above, therefore, this terminal provides all the characters in the XDS 64-character ANSCII set.

XDS STANDARD SYMBOL-CODE CORRESPONDENCES

EBCDIC[†]		Symbol	Card Code	ANSCII[††]	Meaning	Remarks
Hex.	Dec.					
00	0	NUL	12-0-9-8-1	0-0	null	00 through 23 and 2F are control codes.
01	1	SOH	12-9-1	0-1	start of header	
02	2	STX	12-9-2	0-2	start of text	
03	3	ETX	12-9-3	0-3	end of text	
04	4	EOT	12-9-4	0-4	end of transmission	
05	5	HT	12-9-5	0-9	horizontal tab	
06	6	ACK	12-9-6	0-6	acknowledge (positive)	
07	7	BEL	12-9-7	0-7	bell	
08	8	BS or EOM	12-9-8	0-8	backspace or end of message	EOM is used only on XDS Keyboard/ Printers Models 7012, 7020, 8091, and 8092.
09	9	ENQ	12-9-8-1	0-5	enquiry	
0A	10	NAK	12-9-8-2	1-5	negative acknowledge	
0B	11	VT	12-9-8-3	0-11	vertical tab	
0C	12	FF	12-9-8-4	0-12	form feed	
0D	13	CR	12-9-8-5	0-13	carriage return	
0E	14	SO	12-9-8-6	0-14	shift out	
0F	15	SI	12-9-8-7	0-15	shift in	
10	16	DLE	12-11-9-8-1	1-0	data link escape	
11	17	DC1	11-9-1	1-1	device control 1	
12	18	DC2	11-9-2	1-2	device control 2	
13	19	DC3	11-9-3	1-3	device control 3	
14	20	DC4	11-9-4	1-4	device control 4	
15	21	LF or NL	11-9-5	0-10	line feed or new line	
16	22	SYN	11-9-6	1-6	sync	
17	23	ETB	11-9-7	1-7	end of transmission block	
18	24	CAN	11-9-8	1-8	cancel	
19	25	EM	11-9-8-1	1-9	end of medium	
1A	26	SUB	11-9-8-2	1-10	substitute	Replaces characters with parity error.
1B	27	ESC	11-9-8-3	1-11	escape	
1C	28	FS	11-9-8-4	1-12	file separator	
1D	29	GS	11-9-8-5	1-13	group separator	
1E	30	RS	11-9-8-6	1-14	record separator	
1F	31	US	11-9-8-7	1-15	unit separator	
20	32	ds	11-0-9-8-1		digit selector	20 through 23 are used with Sigma EDIT BYTE STRING (EBS) instruction – not input/output control codes.
21	33	ss	0-9-1		significance start	
22	34	fs	0-9-2		field separation	
23	35	si	0-9-3		immediate significance start	
24	36		0-9-4			24 through 2E are unassigned.
25	37		0-9-5			
26	38		0-9-6			
27	39		0-9-7			
28	40		0-9-8			
29	41		0-9-8-1			
2A	42		0-9-8-2			
2B	43		0-9-8-3			
2C	44		0-9-8-4			
2D	45		0-9-8-5			
2E	46		0-9-8-6			
2F	47		0-9-8-7			
30	48		12-11-0-9-8-1			30 through 3F are unassigned.
31	49		9-1			
32	50		9-2			
33	51		9-3			
34	52		9-4			
35	53		9-5			
36	54		9-6			
37	55		9-7			
38	56		9-8			
39	57		9-8-1			
3A	58		9-8-2			
3B	59		9-8-3			
3C	60		9-8-4			
3D	61		9-8-5			
3E	62		9-8-6			
3F	63		9-8-7			

[†] Hexadecimal and decimal notation.

[††] Decimal notation (column-row).

XDS STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC[t]		Symbol	Card Code	ANSCII[tt]	Meaning	Remarks
Hex.	Dec.					
40	64	SP	blank	2-0	blank	
41	65		12-0-9-1			41 through 49 will not be assigned.
42	66		12-0-9-2			
43	67		12-0-9-3			
44	68		12-0-9-4			
45	69		12-0-9-5			
46	70		12-0-9-6			
47	71		12-0-9-7			
48	72		12-0-9-8			
49	73		12-8-1			
4A	74	¢ or ˋ	12-8-2	6-0	cent or accent grave	Accent grave used for left single quote. On model 7670, ˋ not available, and ¢ = ANSCII 5-11.
4B	75	.	12-8-3	2-14	period	
4C	76	<	12-8-4	3-12	less than	
4D	77	(	12-8-5	2-8	left parenthesis	
4E	78	+	12-8-6	2-11	plus	
4F	79	| or ¦	12-8-7	7-12	vertical bar or broken bar	On Model 7670, ¦ not available, and | = ANSCII 2-1.
50	80	&	12	2-6	ampersand	
51	81		12-11-9-1			51 through 59 will not be assigned.
52	82		12-11-9-2			
53	83		12-11-9-3			
54	84		12-11-9-4			
55	85		12-11-9-5			
56	86		12-11-9-6			
57	87		12-11-9-7			
58	88		12-11-9-8			
59	89		11-8-1			
5A	90	!	11-8-2	2-1	exclamation point	On Model 7670, ! is l.
5B	91	$	11-8-3	2-4	dollars	
5C	92	*	11-8-4	2-10	asterisk	
5D	93	)	11-8-5	2-9	right parenthesis	
5E	94	;	11-8-6	3-11	semicolon	
5F	95	~ or ¬	11-8-7	7-14	tilde or logical not	On Model 7670, ~ is not available, and ¬ = ANSCII 5-14.
60	96	-	11	2-13	minus, dash, hyphen	
61	97	/	0-1	2-15	slash	
62	98		11-0-9-2			62 through 69 will not be assigned.
63	99		11-0-9-3			
64	100		11-0-9-4			
65	101		11-0-9-5			
66	102		11-0-9-6			
67	103		11-0-9-7			
68	104		11-0-9-8			
69	105		0-8-1			
6A	106	⌢	12-11	5-14	circumflex	On Model 7670 ⌢ is ¬. On Model 7015 ⌢ is ∧ (caret).
6B	107	,	0-8-3	2-12	comma	
6C	108	%	0-8-4	2-5	percent	
6D	109	_	0-8-5	5-15	underline	Underline is sometimes called "break character"; may be printed along bottom of character line.
6E	110	>	0-8-6	3-14	greater than	
6F	111	?	0-8-7	3-15	question mark	
70	112		12-11-0			70 through 79 will not be assigned.
71	113		12-11-0-9-1			
72	114		12-11-0-9-2			
73	115		12-11-0-9-3			
74	116		12-11-0-9-4			
75	117		12-11-0-9-5			
76	118		12-11-0-9-6			
77	119		12-11-0-9-7			
78	120		12-11-0-9-8			
79	121		8-1			
7A	122	:	8-2	3-10	colon	
7B	123	#	8-3	2-3	number	
7C	124	@	8-4	4-0	at	
7D	125	'	8-5	2-7	apostrophe (right single quote)	
7E	126	=	8-6	3-13	equals	
7F	127	"	8-7	2-2	quotation mark	

[t] Hexadecimal and decimal notation.

[tt] Decimal notation (column-row).

XDS STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC[†]		Symbol	Card Code	ANSCII[††]	Meaning	Remarks
Hex.	Dec.					
80	128		12-0-8-1			80 is unassigned.
81	129	a	12-0-1	6-1		81-89, 91-99, A2-A9 comprise the lowercase alphabet. Available only in XDS standard 89- and 95-character sets.
82	130	b	12-0-2	6-2		
83	131	c	12-0-3	6-3		
84	132	d	12-0-4	6-4		
85	133	e	12-0-5	6-5		
86	134	f	12-0-6	6-6		
87	135	g	12-0-7	6-7		
88	136	h	12-0-8	6-8		
89	137	i	12-0-9	6-9		
8A	138		12-0-8-2			8A through 90 are unassigned.
8B	139		12-0-8-3			
8C	140		12-0-8-4			
8D	141		12-0-8-5			
8E	142		12-0-8-6			
8F	143		12-0-8-7			
90	144		12-11-8-1			
91	145	j	12-11-1	6-10		
92	146	k	12-11-2	6-11		
93	147	l	12-11-3	6-12		
94	148	m	12-11-4	6-13		
95	149	n	12-11-5	6-14		
96	150	o	12-11-6	6-15		
97	151	p	12-11-7	7-0		
98	152	q	12-11-8	7-1		
99	153	r	12-11-9	7-2		
9A	154		12-11-8-2			9A through A1 are unassigned.
9B	155		12-11-8-3			
9C	156		12-11-8-4			
9D	157		12-11-8-5			
9E	158		12-11-8-6			
9F	159		12-11-8-7			
A0	160		11-0-8-1			
A1	161		11-0-1			
A2	162	s	11-0-2	7-3		
A3	163	t	11-0-3	7-4		
A4	164	u	11-0-4	7-5		
A5	165	v	11-0-5	7-6		
A6	166	w	11-0-6	7-7		
A7	167	x	11-0-7	7-8		
A8	168	y	11-0-8	7-9		
A9	169	z	11-0-9	7-10		
AA	170		11-0-8-2			AA through B0 are unassigned.
AB	171		11-0-8-3			
AC	172		11-0-8-4			
AD	173		11-0-8-5			
AE	174		11-0-8-6			
AF	175		11-0-8-7			
B0	176		12-11-0-8-1			
B1	177	\	12-11-0-1	5-12	backslash	
B2	178	{	12-11-0-2	7-11	left brace	
B3	179	}	12-11-0-3	7-13	right brace	
B4	180	[	12-11-0-4	5-11	left bracket	On Model 7670, [is ¢.
B5	181	]	12-11-0-5	5-13	right bracket	On Model 7670,] is !.
B6	182		12-11-0-6			B6 through BF are unassigned.
B7	183		12-11-0-7			
B8	184		12-11-0-8			
B9	185		12-11-0-9			
BA	186		12-11-0-8-2			
BB	187		12-11-0-8-3			
BC	188		12-11-0-8-4			
BD	189		12-11-0-8-5			
BE	190		12-11-0-8-6			
BF	191		12-11-0-8-7			

[†] Hexadecimal and decimal notation.

[††] Decimal notation (column-row).

XDS STANDARD SYMBOL-CODE CORRESPONDENCES (cont.)

EBCDIC†		Symbol	Card Code	ANSCII††	Meaning	Remarks
Hex.	Dec.					
C0	192		12-0			C0 is unassigned.
C1	193	A	12-1	4-1		C1-C9, D1-D9, E2-E9 comprise the uppercase alphabet.
C2	194	B	12-2	4-2		
C3	195	C	12-3	4-3		
C4	196	D	12-4	4-4		
C5	197	E	12-5	4-5		
C6	198	F	12-6	4-6		
C7	199	G	12-7	4-7		
C8	200	H	12-8	4-8		
C9	201	I	12-9	4-9		
CA	202		12-0-9-8-2			CA through CF will not be assigned.
CB	203		12-0-9-8-3			
CC	204		12-0-9-8-4			
CD	205		12-0-9-8-5			
CE	206		12-0-9-8-6			
CF	207		12-0-9-8-7			
D0	208		11-0			D0 is unassigned.
D1	209	J	11-1	4-10		
D2	210	K	11-2	4-11		
D3	211	L	11-3	4-12		
D4	212	M	11-4	4-13		
D5	213	N	11-5	4-14		
D6	214	O	11-6	4-15		
D7	215	P	11-7	5-0		
D8	216	Q	11-8	5-1		
D9	217	R	11-9	5-2		
DA	218		12-11-9-8-2			DA through DF will not be assigned.
DB	219		12-11-9-8-3			
DC	220		12-11-9-8-4			
DD	221		12-11-9-8-5			
DE	222		12-11-9-8-6			
DF	223		12-11-9-8-7			
E0	224		0-8-2			E0, E1 are unassigned.
E1	225		11-0-9-1			
E2	226	S	0-2	5-3		
E3	227	T	0-3	5-4		
E4	228	U	0-4	5-5		
E5	229	V	0-5	5-6		
E6	230	W	0-6	5-7		
E7	231	X	0-7	5-8		
E8	232	Y	0-8	5-9		
E9	233	Z	0-9	5-10		
EA	234		11-0-9-8-2			EA through EF will not be assigned.
EB	235		11-0-9-8-3			
EC	236		11-0-9-8-4			
ED	237		11-0-9-8-5			
EE	238		11-0-9-8-6			
EF	239		11-0-9-8-7			
F0	240	0	0	3-0		
F1	241	1	1	3-1		
F2	242	2	2	3-2		
F3	243	3	3	3-3		
F4	244	4	4	3-4		
F5	245	5	5	3-5		
F6	246	6	6	3-6		
F7	247	7	7	3-7		
F8	248	8	8	3-8		
F9	249	9	9	3-9		
FA	250		12-11-0-9-8-2			FA through FE will not be assigned.
FB	251		12-11-0-9-8-3			
FC	252		12-11-0-9-8-4			
FD	253		12-11-0-9-8-5			
FE	254		12-11-0-9-8-6			
FF	255	DEL	12-11-0-9-8-7		delete	Special – neither graphic nor control symbol.

†Hexadecimal and decimal notation.

††Decimal notation (column-row).

APPENDIX D. ANSCII TO EBCDIC CONVERSION

ANSCII Code		Teletype Character		EBCDIC Character		Echo[4] and Type
Hex.	Octal	Char[1]	Key[2]	Hex.	Prints as[3]	
00	00	(NUL)	P^{cs}	00		0
01	01	(SOH)	A^{c}	01		1
02	02	(STX)	B^{c}	02		1
03	03	(ETX)	C^{c}	03		1
04	04	EOT	D^{c}	04		1
05	05	WRU (ENQ)	E^{c}	09		1
06	06	RU (ACK)	F^{c}	06		1
07	07	BELL (BEL)	G^{c}	07	bell	# +1
08	10	(BS)	H^{c}	08		1
09	11	TAB (HT)	I^{c}	05		3
0A	12	LF	LINE FEED	25	line feed	@+7
0B	13	VT	K^{c}	0B		1
0C	14	FORM (FF)	L^{c}	0C		@+1
0D	15	CR	RETURN	15	carriage return	@+7
0E	16	(SO)	N^{c}	0E		1
0F	17	(SI)	O^{c}	0F		1
10	20	(DLE)	P^{c}	10		1
11	21	(DC1)	Q^{c}	11		1
12	22	TAPE (DC2)	R^{c}	12		1
13	23	X-OFF (DC3)	S^{c}	13		1
14	24	(DC4)	T^{c}	14		1
15	25	(NAK)	U^{c}	0A		1
16	26	(SYN)	V^{c}	16		1
17	27	(ETB)	W^{c}	17		1
18	30	(CAN)	X^{c}	18		1
19	31	(EM)	Y^{c}	19		1
1A	32	(SS)	Z^{c}	1A		1
1B	33	(ESC)	K^{cs}	1B		1
1C	34	(FS)	L^{cs}	1C		1
1D	35	(GS)	M^{cs}	1D		1
1E	36	(RS)	N^{cs}	1E		1
1F	37	(US)	O^{cs}	1F		1
20	40	blank	SPACE BAR	40	blank	#+ 1
21	41	!	1^{s}	5A	(!)	#+ 2
22	42	"	2^{s}	7F	(")	#+ 2
23	43	#	3^{s}	7B	#	#+ 1
24	44	$	4^{s}	5B	$	#+ 1
25	45	%	5^{s}	6C	%	#+ 2
26	46	&	6^{s}	50	&	#+ 2
27	47	'	7^{s}	7D	'	#+ 2
28	50	(	8^{s}	4D	(	#+ 2
29	51	)	9^{s}	5D	)	#+ 2
2A	52	*	$:^{s}$	5C	*	#+ 2
2B	53	+	$;^{s}$	4E	+	#+ 2
2C	54	,	,	6B	,	#+ 2
2D	55	-	-	60	-	#+ 2
2E	56	.	.	4B	.	#+ 2
2F	57	/	/	61	/	#+ 3
30	60	0	0	F0	0	#+ 1
31	61	1	1	F1	1	#+ 1
32	62	2	2	F2	2	#+ 1
33	63	3	3	F3	3	#+ 1
34	64	4	4	F4	4	#+ 1
35	65	5	5	F5	5	#+ 1
36	66	6	6	F6	6	#+ 1
37	67	7	7	F7	7	#+ 1
38	70	8	8	F8	8	#+ 1
39	71	9	9	F9	9	#+ 1
3A	72	:	:	7A	:	#+ 1
3B	73	;	;	5E	;	#+ 2
3C	74	<	$,^{s}$	4C	<	#+ 2
3D	75	=	$-^{s}$	7E	=	#+ 2
3E	76	>	$.^{s}$	6E	>	#+ 2
3F	77	?	$/^{s}$	6F	(?)	#+ 2
40	100	@	P^{s}	7C	@	#+ 1

ANSCII Code		Teletype Character		EBCDIC Character		Echo[4] and Type
Hex.	Octal	Char[1]	Key[2]	Hex.	Prints as[3]	
41	101	A	A	C1	A	#+ 1
42	102	B	B	C2	B	#+ 1
43	103	C	C	C3	C	#+ 1
44	104	D	D	C4	D	#+ 1
45	105	E	E	C5	E	#+ 1
46	106	F	F	C6	F	#+ 1
47	107	G	G	C7	G	#+ 1
48	110	H	H	C8	H	#+ 1
49	111	I	I	C9	I	#+ 1
4A	112	J	J	D1	J	#+ 1
4B	113	K	K	D2	K	#+ 1
4C	114	L	L	D3	L	#+ 1
4D	115	M	M	D4	M	#+ 1
4E	116	N	N	D5	N	#+ 1
4F	117	O	O	D6	O	#+ 1
50	120	P	P	D7	P	#+ 1
51	121	Q	Q	D8	Q	#+ 1
52	122	R	R	D9	R	#+ 1
53	123	S	S	E2	S	#+ 1
54	124	T	T	E3	T	#+ 1
55	125	U	U	E4	U	#+ 1
56	126	V	V	E5	V	#+ 1
57	127	W	W	E6	W	#+ 1
58	130	X	X	E7	X	#+ 1
59	131	Y	Y	E8	Y	#+ 1
5A	132	Z	Z	E9	Z	#+ 1
5B	133	{[} (|)	K^{s}	4F	|	#+ 2
5C	134	{\\} (\\)	L^{s}	4A	(¢)	#+ 2
5D	135	{]} (¬)	M^{s}	5F	(¬)	#+ 2
5E	136	↑ (^)	N^{s}	6A		#+ 3
5F	137	← (-)	O^{s}	6D	(-)	#+ 2
⋮	⋮	⋮	⋮	⋮	⋮	0
⋮	⋮	⋮	⋮	⋮	⋮	
7E	176	ESC	ESCAPE	1B		0
7F	177	DEL	RUBOUT	FF		0

Notes:

1. The forms in parentheses appear only on XDS Teletype #7015, whereas the unparenthesized forms appear on the specified keys of all standard model Teletypes. (Some models lack the ESCAPE key). The forms in braces print when the specified key is depressed, but they do not appear on the keys.
2. Superscript c indicates use of the CTRL key; superscript s indicates use of the SHIFT key.
3. The forms in parentheses are contained in the XDS 63- and 89-graphic character sets but not in the standard 57-graphic character set. On printers equipped with only the standard character set, these forms will print as blanks.
4. The echo and type specifies the echoability and activation type of a character.

The Teletype character mnemonics have the following meanings:

ACK	Acknowledge	ENQ	Enquire	NAK	Negative acknowledge
BEL	Bell	EM	End of medium	NUL	Null
BS	Backspace	EOT	End of transmission	RS	Record separator
CAN	Cancel	ESC	Escape	SI	Shift in
CR	Carriage return	ETB	End of transmission block	SO	Shift out
DC1	Device control 1	ETX	End of text	SOH	Start of header
DC2	Device control 2	FF	Form feed	SS	Start of special sequence
DC3	Device control 3	FS	File separator	STX	Start of text
DC4	Device control 4	GS	Group separator	SYN	Synchronize
DEL	Delete	HT	Horizontal tab	US	Unit separator
DLE	Data link escape	LF	Line feed	VT	Vertical tab

APPENDIX E. EBCDIC TO ANSCII CONVERSION

EBCDIC Code		Teletype Character		ANSCII Code	
Hex.	Prints as[3]	Char[1]	Key[2]	Hex.	Octal
00		(NUL)	P^{cs}	00	00
01		(SOH)	A^{c}	01	01
02		(STX)	B^{c}	02	02
03		(ETX)	C^{c}	03	03
04		EOT	D^{c}	04	04
05		TAB (HT)	I^{c}	09	11
06		RU (ACK)	F^{c}	06	06
07	bell	BELL (BEL)	G^{c}	07	07
08		(BS)	H^{c}	08	10
09		WRU (ENQ)	E^{c}	05	05
0A		(NAK)	U^{c}	15	25
0B		VT	K^{c}	0B	13
0C		FORM (FF)	L^{c}	0C	14
0D	carriage return	CR	RETURN	0D	15
0E		(SO)	N^{c}	0E	16
0F		(SI)	O^{c}	0F	17
10		(DLE)	P^{c}	10	20
11		(DC1)	Q^{c}	11	21
12		TAPE (DC2)	R^{c}	12	22
13		X-OFF (DC3)	S^{c}	13	23
14		(DC4)	T^{c}	14	24
15	carriage return	CR	M^{c} or RETURN	0D	15
16		(SYN)	V^{c}	16	26
17		(ETB)	W^{c}	17	27
18		(CAN)	X^{c}	18	30
19		(EM)	Y^{c}	19	31
1A		(SS)	Z^{c}	1A	32
1B		(ESC)	K^{cs}	1B	33
1C		(FS)	L^{cs}	1C	34
1D		(GS)	M^{cs}	1D	35
1E		(RS)	N^{cs}	1E	36
1F		(US)	O^{cs}	1F	37
⋮	⋮	⋮	⋮	⋮	⋮
25	line feed	LF	J^{c} or LINE FEED	0A	12
⋮	⋮	⋮	⋮	⋮	⋮
40	blank	blank	SPACE BAR	20	40
⋮	⋮	⋮	⋮	⋮	⋮
4A	(¢)	{\} (\)	L^{s}	5C	134
4B	.	.	.	2E	56
4C	<	<	$,^{s}$	3C	74
4D	(	(	8^{s}	28	50
4E	+	+	$;^{s}$	2B	53
4F	|	{[} (|)	K^{s}	5B	133
50	&	&	6^{s}	26	46
⋮	⋮	⋮	⋮	⋮	⋮
5A	(!)	!	1^{s}	21	41
5B	$	$	4^{s}	24	44
5C	*	*	$:^{s}$	2A	52
5D	)	)	9^{s}	29	51
5E	;	;	;	3B	73
5F	(¬)	{]} (¬)	M^{s}	5D	135
60	-	-	-	2D	55
61	/	/	/	2F	57
⋮	⋮	⋮	⋮	⋮	⋮
6A		↑ (^)	N^{s}	5E	136
6B	,	,	,	2C	54
6C	%	%	5^{s}	25	45

EBCDIC Code		Teletype Character		ANSCII Code	
Hex.	Prints as[3]	Char[1]	Key[2]	Hex.	Octal
6D	(-)	← (-)	0^s	5F	137
6E	>	>	$.^s$	3E	76
6F	(?)	?	$/^s$	3F	77
⋮	⋮	⋮	⋮	⋮	⋮
7A	:	:	:	3A	72
7B	#	#	3^s	23	43
7C	@	@	P^s	40	100
7D	'	'	7^s	27	47
7E	=	=	$-^s$	3D	75
7F	(")	"	2^s	22	42
⋮	⋮	⋮	⋮	⋮	⋮
C1	A	A	A	41	101
C2	B	B	B	42	102
C3	C	C	C	43	103
C4	D	D	D	44	104
C5	E	E	E	45	105
C6	F	F	F	46	106
C7	G	G	G	47	107
C8	H	H	H	48	110
C9	I	I	I	49	111
⋮	⋮	⋮	⋮	⋮	⋮
D1	J	J	J	4A	112
D2	K	K	K	4B	113
D3	L	L	L	4C	114
D4	M	M	M	4D	115
D5	N	N	N	4E	116
D6	O	O	O	4F	117
D7	P	P	P	50	120
D8	Q	Q	Q	51	121
D9	R	R	R	52	122
⋮	⋮	⋮	⋮	⋮	⋮
E2	S	S	S	53	123
E3	T	T	T	54	124
E4	U	U	U	55	125
E5	V	V	V	56	126
E6	W	W	W	57	127
E7	X	X	X	58	130
E8	Y	Y	Y	59	131
E9	Z	Z	Z	5A	132
⋮	⋮	⋮	⋮	⋮	⋮
F0	0	0	0	30	60
F1	1	1	1	31	61
F2	2	2	2	32	62
F3	3	3	3	33	63
F4	4	4	4	34	64
F5	5	5	5	35	65
F6	6	6	6	36	66
F7	7	7	7	37	67
F8	8	8	8	38	70
F9	9	9	9	39	71
⋮	⋮	⋮	⋮	⋮	⋮
FF		CD	RUBOUT	7F	177

Codes X'81' through X'A9', which represent the lower case letters, are exactly congruent to X'C1' through X'E9' and may be used interchangeably. All will print as capital letters on standard Teletypes.

Notes:

1. The forms in parentheses appear only on XDS Teletype #7015, whereas the unparenthesized forms appear on the specified keys of all standard model Teletypes. (Some models lack the ESCAPE key.) The forms shown in braces print when the specified key is depressed, but they do not appear on the keys.

2. Superscript c indicates use of the CTRL key; superscript s indicates use of the SHIFT key.

3. The forms in parentheses are contained in the XDS 63- and 89-graphic-character sets but not in the standard 57-graphic-character set. On printers equipped with only the standard character set, these forms will print as blanks.

APPENDIX F. BPM/BTM MONITOR SIZING

INTRODUCTION

The information and formulas in this appendix may be used by the systems analyst to determine the core requirements and system RAD and/or disk pack PSA requirements for any BPM/BTM Monitor (F01 version only). The information required to determine BLL and PSA size consists of all PASS2 information and the Monitor tree structure.

PSA size is dependent upon the Monitor root size, which in turn, is dependent on HGP size. HGP size will vary according to PFA/PER storage. PFA/PER storage available on the systems device is dependent upon the amount of PSA area required. Therefore, one must approximate the PSA size and then proceed with the Monitor root size calculations. The actual PSA size may then be calculated. Changing the approximated PSA size to the actual size required will vary the Monitor root size by only a few words at most and should not affect the actual PSA size just calculated (brief check will confirm this). A figure of X'18' as an approximate PSA size is quite reasonable for a real-time BTM system on a 7232.

MONITOR CORE REQUIREMENTS

The amount of core required by different Monitors and different configurations will vary only in the root segment of the Monitor. Provided two overlays are used, the Monitor overlay area will usually require A5C words. The size of the root of the Monitor must be calculated by summing the lengths of the programmed modules placed in the root of the system and then adding the length of each PASS2-generated table. The programmed modules and their lengths are listed in Table F-1.

Notice that HANDLERS is included. The size given is the size of BASHANDL, the basic I/O handlers that must be present. Any additional handlers required must also be added in. BASHANDL contains handlers for the CR, TY, LP, RAD, 9T, 7T and Plotter devices. Table F-2 lists the size of the remaining available handlers.

The size of the PASS2-generated modules listed below must be calculated according to PASS2 parameters if they are present in the system. The ones denoted as BPM are necessary. The remaining modules are for BTM, R/T, or symbiont systems only.

PASS2 Modules	System
MON : : ORG	BPM
ROOT	BPM
M : CPU	BPM
M : ABS	BPM
M : SDEV	Symbiont only
M : BTM	BTM
M : JIT	BPM (but zero length)
M : FRGD	R/T only
IOTABLE	BPM

PASS2 MODULE CALCULATIONS

The formulas that follow describe how to calculate the length of each of the PASS2-generated modules listed previously. All values given are in decimal and all calculations are intended to be performed in decimal. Brackets indicate that the integer portion of the results of the enclosed terms is to be used.

MON : : ORG is the value specified by the ORG parameter on the :MONITOR card.

ROOT is nearly always a constant 84 words. It will be two words less for each segment removed from the Monitor tree structure. Thus, for example, if ALTCP were moved to the root then ROOT would be 82 words long.

M : CPU is calculated as follows:

$$\text{TSTACK}+34(\text{MPOOL}-2)+256(\text{SPOOL})+19(\text{CFU}+2)+2(\text{SFIL})$$

$$+40(\text{CPOOL})+\text{MPATCH}+\left[\frac{\text{core size in K}+7}{8}\right]$$

$$+9\left[\frac{Q+3}{4}\right]+\left[\frac{Q+1}{2}\right]+5Q+172$$

where Q = QUEUE size.

M : ABS is calculated by using the formula below to determine the core requirements for each ABS'd processor. Five words must be added to the sum of the calculated lengths.

$$\left(\left[\frac{\text{name length}+4}{4}\right]+4\right) \text{ for each processor}$$

M : SDEV is calculated as follows:

$$6\left[\frac{n+1}{4}\right]+\left[\frac{n+1}{2}\right]+3n+16$$

where n is the number of symbiont devices. Each RBT should be counted as three symbiont devices.

IOTABLE is calculated as follows:

$$9\left[\frac{n+4}{4}\right]+4\left[\frac{n+2}{2}\right]+8(n+1)+4\left[\frac{c+3}{4}\right]+c$$

$$+3(\#\text{tapes}+\#\text{DPs})+n+12(\#\text{DPs})+8(\#\text{tapes})$$

$$+2(\#\text{CRs})+74(\#\text{CPs})+6(n-\#\text{DPs}-\#\text{tapes}-\#\text{CRs}-$$

$$\#\text{CPs})+53$$

where

n is the number of devices (i.e., number of :DEVICE cards, including the COC device and counting an RBT as three devices).

c is the number of logical channels, i.e., number of :CHAN cards.

The formula for IOTABLE size excludes HGPs. HGP sizes may be calculated as shown below. Use the appropriate formula for each RAD and/or disk pack, sum the results, and add it to the results of the above formula for IOTABLE. In each of the formulas below, an extra word is included, since all HGPs start on a doubleword boundary.

SWAPPER 8 words

Any RAD or disk pack used as a swapper only (no PFA/PER) will require eight words for an HGP.

7204

$$\left[\left(\left[\frac{(\#\text{PFA/PER trks})\ 16}{6}\right]+31\right)\div 32\right]+8$$

or 51 words if all tracks are PFA/PER.

7232

$$\left[\frac{(\#\text{PFA/PER trks})6+31}{32}\right]+8$$

or 104 words if all tracks are PFA/PER.

7212

$$\left[\frac{(\#\text{PFA/PER trks})41+31)}{32}\right]+8$$

or 90 words if all tracks are PFA/PER.

7242

$$\left[\frac{(\#\text{PFA/PER trks})3+31}{32}\right]+8$$

or 383 words if all tracks are PFA/PER, or 21 words if Cylinder Allocation is used (note that PRIVATE invokes Cylinder Allocation).

M : BTM is calculated as follows:

$$\left[\frac{NU*OB+3}{4}\right]+\left[\frac{NU*IB+3}{4}\right]+14\left[\frac{NU+3}{4}\right]+37\left[\frac{NU+1}{2}\right]$$

$$+7NU+\left[\frac{2NU+5}{4}\right]+\left[\frac{NS+3}{4}\right]+8\left[\frac{NS+1}{2}\right]$$

$$+2NS+SWPLIST+28$$

where

NU is the number of users.

NS is the number of subsystems.

OB is the output buffer size in bytes.

IB is the input buffer size in bytes.

SWPLIST equals 30 for 7232/7212/7242 swapper; or equals 6 *(# pages user area) for 7204 swapper.

Note that SWPLIST is constant regardless of the number of swapping RADs and/or disk packs used.

If the BTM Performance Monitor is included into the system, the results of the following formula must be added to the results of the formula above for M : BTM (this is for tables only). The Performance Monitor is itself added into HANDLERS in addition to the amount below (see Table F-2):

3NU+4NS+2NUP+178

where

NU, NS are the same as above.

NUP is the number of user pages.

Note that the number of user pages in this formula and the one above it must include the 4 pages of context area, i.e., USERSIZE in pages +4.

M : FRGD is calculated as follows:

$$12(\text{NFRGD})+12(\text{NINT})+4(\text{CTQ})+4(\#\text{INTS})$$

$$+2\left[\frac{\#\text{INT LABELS}+2}{2}\right]+\left[\frac{\text{CORE SIZE IN K}+7}{8}\right]+200$$

PSA SIZE REQUIREMENTS

The PSA sizing chart shown here is presented in the steps that the Monitor uses in allocating the PSA during PASS0. For each step a sector boundary is required. Therefore, the number of sectors required may be calculated by dividing the number of words by 256 for a 7252, 7212, 7242 device, or by 90 for a 7204, and then adding one if the remainder is nonzero. If a DP is used as the system device, any one step may not cross a cylinder boundary. A cylinder contains 120 decimal sectors. Thus, as many as 119 sectors could be left unused between two steps. Also, when a DP is used, each allocation step must use contiguous non-flawed sectors. The steps are as follows:

Step	PSA Allocation
1	One sector for BOOTSTRAP
2	All HGPs + 3 words
3	2(SFIL+1)+2
4	19(CFU+2)+1
5	Monitor overlay segment 0
6	Monitor overlay segment 1
⋮	
N	Last Monitor overlay segment
N+1	Monitor root
N+2	36 sectors for DCBs
N+3	ABSGOSZ or 512 words, whichever is greater. ABSGOSZ is specified on the :ABS card.
N+4	ABS processor #1 00 section
N+5	ABS processor #1 01 section
N+6	ABS processor #2 00 section
⋮	
N+M	Last ABS processor 01 section

The list of sizes in Table F-3 may be used for determining the amount of PSA area required for ABSed processors.

Table F-1. Monitor Module Sizes

			2662 Words (A5C Hex.)					
ROOT Seg.	Hex.	Dec.	Overlay 1	Hex.	Dec.	Overlay 2	Hex.	Dec.
ENTRY	3FA	1018	PRGMLDR	76A	1898	OPNL-OBSE	416	1046
SIMINT	60	96	TYPR	322	802	M:1E-OBSE	14E	334
DECSIM	208	520	IOD	104	260	OPN-OBSE	402	1026
FLTSIM	E6	230	BTMNRES	410	1040	CLS----		
BYTSIM	62	98	DEBUG-DUMP	2BA	698	MUL	2CE	718
CVTSIM	34	52	EXIT	49E	1182	SEGLOAD-OBSE	364	868
IOSYM	402	1026	M:15	1F6	502	WRTF	436	1078
IO	3CC	972	M:16	F6	246	WRTD-CCLOSE	2B4	692
PFSR	96	150	M:17	FA	250	LBLT	22C	556
FBCD	2C	44	KEYIN1	34E	846	M:19	2C8	712
RTROOT	9E0	2528	KEYIN2	5BC	1468	ALTCP[tt]	11A	282
HANDLERS[t]	440	1088	M:18	1E8	488	POS	2C4	708
COOP	1FC	508	M:1A	4DC	1244			
CALPROC	9A	154	JOBENT	29C	668			
IORT	386	902	MEMALOC[tt]	1CE	462			
TOPRT	0	0	RDF	626	1574			
			RCVR	84E	2126			
			RCVR2	6D6	1750			
			LDPRG	3D8	984			

[t] See Table F-2.

[tt] Should be in the ROOT in a R/T system if possible.

Table F-2. I/O Handler Sizes

Handlers	Hex.	Dec.
BASHANDL	440	1088
CRDOUT	4E	78
DPAK	7E	126
COC	1300	4864
RBT	4B0	1200
BTMPM	E6	230
PTAP	96	150

Table F-3. Processor Sizes

	00 Size		01 Size	
Processor	Hex.	Dec.	Hex.	Dec.
BPM BASIC	1CFA	7418	118	280
BPM COBOL[†]	56C	1388	5D6	1494
BPM SYMBOL	1E0	480	ECA	3786
CCI	2570	9584	480	1168
DEF[†]	5E0	1504	258	600
DEFCOM	4D6	1238	B2	178
DMS				
DMSDUMP	10BA	4282	176	374
DMSINIT	C2A	3114	12E	302
DMSLOAD	1878	6264	1C0	448
FDP[†]	EF4	3828	10D6	4310
EDCON	30E	782	810	2064
ELIST	C56	3158	AA	170
ERRWRT	9E	158	E6	230
FMGE	766	1894	36E	878
FORTRAN IV-H	1FBC	8124	2F8	760
FORTRAN IV[†]	652	1618	F02	3842
FPURGE	906	2310	2AA	682
LOADER	1E6	486	1CEC	7404
LOCCT	444	1092	14C	332
LOPE	278	632	9F4	2548
MANAGE				
DICTNARY[†]	26A	618	210	528
FILE UP[†]	24C	588	448	1096

[†] Denotes an overlaid processor. Note that only the size of the root is given for overlaid processors since only the root portion is ABSed.

Table F-3. Processor Sizes (cont.)

Processor	00 Size		01 Size	
	Hex.	Dec.	Hex.	Dec.
REPORT	318	792	DC2	3522
RETRIEVE†	6C8	1736	3AE	942
MEDDUMP	28B0	10416	E4	228
MERGE	41C	1052	8D0	2256
META-SYMBOL†	28E	654	B4A	2890
MONDUMP†	2B7E	11134	1F0	496
OLAY†	1E6	486	2B6	694
PASS1†	89E	2206	2A2	674
PASS2†	13AE	5038	196	406
PASS3	E26	3622	17E	382
PCL	264	612	FB2	4018
PFIL	324	804	756	1878
REW	324	804	756	1878
ROMTRAN	1DC	476	9F0	2544
SORT†	B18	2840	3E0	992
SUPER	B0C	2828	E6	230
VOLINIT	1660	5728	98	152
WEOF	324	804	756	1878

†Denotes an overlaid processor. Note that only the size of the root is given for overlaid processors since only the root portion is ABSed.

APPENDIX G. REAL-TIME RESPONSE TIME

Response time for real-time tasks is the time elapsed before entry to the task once the task's level is the highest priority level in the "wait" state provided no levels of higher priority advance to "active" during this period and provided no other external level has either inhibited the external interrupts or disarmed or disabled the task's level. These two provisions are controllable by the user only and are his responsibility. Thus, response time is a function of Monitor overhead only.

Total response time consists of two parts:

1. The time required for the level to advance from "wait" to "active" state.

2. The time elapsed after advancing to "active" state until entry to the task.

Once the external level is in "wait" state, it may be prevented from advancing to "active" state only by either a higher level in "wait" or "active", or by the external interrupts being inhibited. Thus, this time is either the amount of time that the Monitor has a clock or I/O level active or the amount of time that the Monitor inhibits external interrupts. Once the level is "active", the remaining response time is due to Monitor execution and I/O necessary to control the machine environment. Not all tasks are affected by each of these two types of response time.

There are basically three types of real-time tasks:

1. Directly connected.
2. Centrally connected resident.
3. Centrally connected nonresident.

Directly connected tasks are subject to the first type of response time only. All centrally connected tasks are subject to both types of response time. So, for total response time, one should first measure type 1, then add the type 2 time for centrally connected tasks.

The amount of time that clocks and/or the I/O level is active and the time that the external interrupts are inhibited, will vary considerably between system configurations and the load on the system at the time of the measurement. External interrupts are inhibited each time the Monitor enters code that is not reentrant. The frequency of this is dependent on the amount of Monitor services being performed, which is determined by the load on the system (as is I/O interrupt servicing). Servicing I/O interrupts requires a table look-up to find the interrupting device, and the I/O level is not cleared until the device is identified. So this time is dependent on the number of devices on the system.

Once the level advances to the "active" state, the directly connected task is entered immediately via the XPSD which is executed when the level advances to "active" state. Centrally connected tasks, however, are entered only after the Monitor switches the machine environment. For core resident tasks, this takes approximately 170 microseconds. Thus, the response time would be the time required from "wait" to "active" state (type 1) plus 170 microseconds.

Centrally connected tasks that are RAD and/or disk pack resident require the above time plus the time to read them from the RAD or disk pack. Reading tasks from the RAD and/or disk pack is not performed by the file I/O routines but it is done by code designed just for reading real-time tasks and checkpointing the background. Thus, it is much faster than file I/O. However, the time required is subject to the IOP and RAD and/or disk pack type, and, of course, current I/O load on that RAD or disk pack. The read is done via QUEUE and at the priority of the nonresident task. This results, though, in a greater length of time before control is given to the task during which another clock, I/O, and/or external interrupt may be serviced. Also, any higher priority I/O or I/O clean-up to the RAD or disk pack that had been deferred will be forced to completion before the task is read in.

A minimum of one read for each control section is required to bring the task into core. The total number of reads required depends on the task size and the type of device. For each read the record size is the control section size or a maximum record size for the particular device, whichever is smaller. Note that each control section is queued separately. The maximum record sizes are, in hexadecimal words, as follows:

7204	FFF0
7212	10000
7232	10000
7242	1800

If the nonresident task is biased in the background then all nonsymbiont background I/O is first run to completion. (This includes all nonsymbiont I/O that is queued.) Then the background area of core is written to the RAD and/or disk pack as described above for nonresident tasks. It is written as if it were one control section. So, for nonresident background biased tasks, this time for saving the background must be added to "wait", to "active" state time, the machine environment switching time, and the task read time. If, however, at the time the level goes to "active" state the background is already saved, this function is not performed and the response time is the same as for nonresident centrally connected tasks biased in the foreground.

The function of saving the background is done by the control task at the control task's level. A request for a background save is queued for the control task, the control task level is triggered, and the nonresident task's level is cleared. When the control task has completed the background save, it will trigger the task's level again. This time the background is already saved and task is read just as a centrally connected, nonresident, foreground biased task. But once the control task triggers the task's level, all of the response timings are again applicable. Also, since the control task does the background save, the nonresident task may have to wait for all other tasks (including lower priority tasks) to clear their level before it gains control.

APPENDIX H. LABELED TAPE SENTINELS

The formats of sentinels for labeled tapes are described below. All sentinels begin on a word boundary (see Figure H-1).

:LBL

This record identifies the reel number of the tape. Reel numbers are four alphanumeric characters in length. Sentinel length: 12 bytes (see Figure H-2).

:ACN

This sentinel identifies the owner of the tape, the expiration date, and the creation date, in that order.

The account number is 8 alphanumeric characters in length, left-justified and in EBCDIC code (see Figure H-3).

The dates are of the form $m_1m_2d_1d_2bby_1y_2$, where m_1m_2 is the numerical representation of the month, d_1d_2 the day, bb are blanks, and y_1y_2 are the last two digits of the year. The digits are in EBCDIC and the blanks must appear.

Sentinel length: 28 bytes followed by a physical end-of-file (tape mark record).

:BOF

The beginning-of-file sentinel consists of the the file information record, the user's label (if the user has specified one) and a physical end-of-file. The file information consists of control words and the information itself (see Figures H-4 and H-5). A control word has the following form:

Code	LEI	Length
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

1. Code identifies the type of information following the control word.

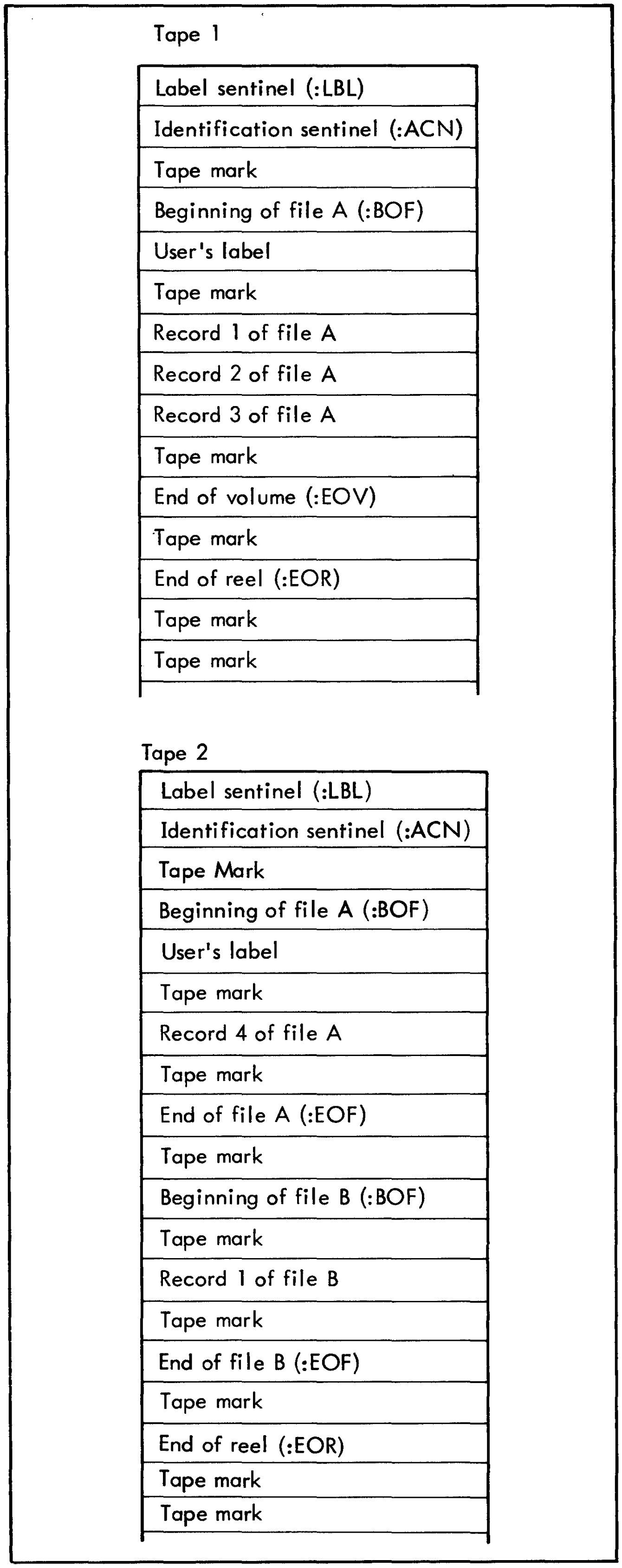
   The codes are:

   01 - file name. The file name may be a maximum of 31 characters. An additional byte is used to state the length of the file name.

   03 - password (2 words, left-justified).

   05 - READ account numbers.

   06 - WRITE account numbers. Each account number is left-justified, blank-filled, and two words long. The total number of READ and WRITE accounts must not exceed 16. READ accounts identify those who may have only read access to the file. WRITE accounts identify those who may read and write the file. NONE or ALL are also allowed.



Tape 1

Label sentinel (:LBL)
Identification sentinel (:ACN)
Tape mark
Beginning of file A (:BOF)
User's label
Tape mark
Record 1 of file A
Record 2 of file A
Record 3 of file A
Tape mark
End of volume (:EOV)
Tape mark
End of reel (:EOR)
Tape mark
Tape mark

Tape 2

Label sentinel (:LBL)
Identification sentinel (:ACN)
Tape Mark
Beginning of file A (:BOF)
User's label
Tape mark
Record 4 of file A
Tape mark
End of file A (:EOF)
Tape mark
Beginning of file B (:BOF)
Tape mark
Record 1 of file B
Tape mark
End of file B (:EOF)
Tape mark
End of reel (:EOR)
Tape mark
Tape mark

Figure H-1. General Format of Labeled Tape

:	L	B	L
x	x	x	x
x	x	x	x

Figure H-2. Label Sentinel

:	A	C	N
a_1	a_2	a_3	a_4
a_5	a_6	a_7	a_8
m_1	m_2	d_1	d_2
ƀ	ƀ	y_1	y_2
m_1	m_2	d_1	d_2
ƀ	ƀ	y_1	y_2
Inter-record gap			
Tape mark record			

Figure H-3. Identification Sentinel

:	B	O	F
File information (see Figure D-7)			
Inter-record gap			
User's label			
Inter-record gap			
Tape mark record			

Figure H-4. Beginning-of-File Sentinel

:	B	O	F
Code 01	Last entry indicator 00	▨	Length
No. char. in file name	File name		
03	00	▨	Length
Password			
05	00	▨	Length
READ	Account numbers		
06	00	▨	Length
WRITE	Account numbers		
09	01	▨	Length
ORG	KEYM	VOL	▨
HDL	▨	▨	▨
0 7	8 15	16 23	24 31

Figure H-5. File Information on Tape

09 - Miscellaneous information, such as:

ORG – gives the file organization, which may be keyed or consecutive.

KEYM – specifies the maximum length of the keys. Keys may not be greater than 31 bytes. An additional byte is used to specify the length of the key. On consecutive files, the length of the dummy key is assumed to be three, therefore, KEYM is ignored. On keyed files, if KEYM = 0, the maximum length is assumed to be 11.

VOL – On multi-reel files, this entry specifies the position of this tape in the file. For example, VOL = 2 implies this is the second tape of the multi-reel file. Every file begins with VOL = 1 (including single-reel files).

HDL – This specifies the length of the user's label. If HDL = 0, then no user's label exists and the following record must be a physical end-of-file.

2. LEI is the last-entry indicator; this entry in the control word indicates the end of the file information. The control words, along with the information they define, do not have to be in a particular order, but LEI must equal 0 if the file information entry is not the last one and must equal 1 if the entry is the last one.

3. Length specifies the length, in words, of the information associated with a particular entry (i.e., following the code word).

:EOF, :EOV, and :EOR

These sentinels are described in Figures H-6, H-7, and H-8. The notation "PBS" represents the value of the previous block size, in bytes.

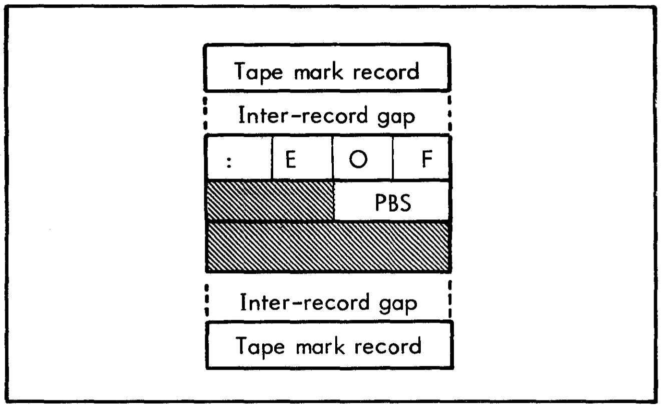


Figure H-6. End-of-File Sentinel

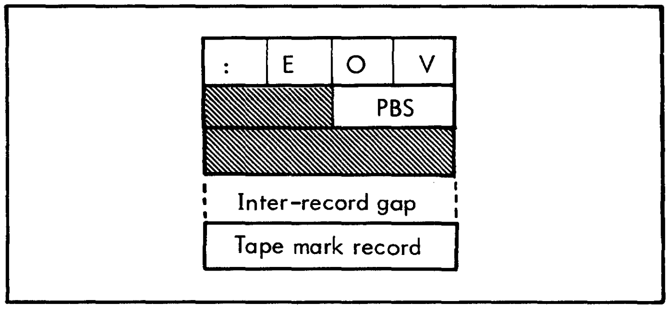


Figure H-7. End-of-Volume Sentinel

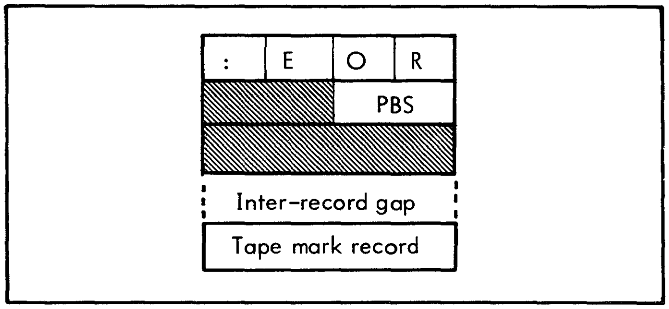


Figure H-8. End-of-Reel Sentinel

INDEX

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Note: For each entry in this index, the number of the most significant page is listed first. Any pages thereafter are listed in numerical sequence.

Xerox Data Systems

READER COMMENT FORM

XEROX®

We would appreciate your comments and suggestions for improving this publication.

Publication No.	Rev. Letter	Title	Current Date

How did you use this publication?

□ Learning □ Installing □ Operating
□ Reference □ Maintaining □ Sales

Is the material presented effectively?

□ Fully covered □ Well illustrated
□ Clear □ Well organized

What is your overall rating of this publication?

□ Very good □ Fair □ Very poor
□ Good □ Poor

What is your occupation?

Your other comments may be entered here. Please be specific and give page, column, and line number references where applicable. To report errors, please use the XDS Software Improvement or Difficulty Report (1188) instead of this form.

Thank you for your interest.

Fold and fasten as shown on back.
No postage needed if mailed in U.S.A.

Your name and return address.

STAPLE

STAPLE

FOLD

FIRST CLASS
PERMIT NO. 229
EL SEGUNDO, CALIF.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY

Xerox Data Systems

701 South Aviation Boulevard
El Segundo, California 90245

ATTN: PROGRAMMING PUBLICATIONS

CUT ALONG LINE

FOLD