PART I

DESIGN DIRECTION

TABLE OF CONTENTS

0.0 INTRODUCTION

1.0 INTRODUCTION

This part of the NOS/VE Design Specification describes the basic design direction. Basic concepts are described and for each major function the design objectives, assumptions and constraints of the design, design approach and major NOS/170 differences impacting end-user interfaces are described.

2.0 CONCEPTS

2.1 JOB

A job is the vehicle through which users interface NOS/VE. Users may present jobs to the operating system from either batch input devices or from interactive terminals. In the first case, a complete sequence of command language statements is transferred from the input device to system mass storage for subsequent processing. In the second case, command language statements are interpreted a statement at a time as they are read from the terminal.

A job is comprised of a user defined environment and a system defined environment. The user defined environment consists of system command language statements and variables which describe and direct the user's computing process. The system defined environment consists of a set of system oriented structures which permit the operating system to uniquely identify, monitor and account for the user computing session.

Important aspects of a job include:

- Access control
  - Each job runs on behalf of a single validated user; a user is identified by (<family name> <user name> <project name> <account name>).
  - User identity is validated at the time the job is initiated and relied upon for the duration of the job.
  - The user validation process determines a definition of the job's restrictions and capabilities that is inforced during job execution.
  - A system security level governs the validation process in determining user access and the type of programs, resources and data that may be used.
- Job elements

2.0 CONCEPTS
2.1 JOB

- Every job consists of multiple tasks. A task is the only asynchronous component of a job.

- Jobs are broken down into tasks for protection, serialization, multiprogramming and multiprocessing purposes.

- Every job has a controlling system task responsible for orderly termination of the job.

Scheduling

- All batch jobs are queued prior to entry into the system and selected for execution based on user supplied priority, class and mode.

- Batch and interactive jobs are preemptively scheduled. The scheduler is parameterized to allow sites to tune their systems to their needs.

Multiple sources

- Users may submit a job from any terminal or host in a network to any other host using a logical identification parameter (family name).

- Job output is automatically returned to the host system and submitting terminal.

- Jobs may be submitted from executing jobs. This process allows a basic job sequencing capability.

Resource allocation

- Resources are allocated and deallocated dynamically during job execution.

- Resource allocation limits are determined during validation on a user/account/project basis. These limits may be further constrained explicitly by job commands and requests.

Logging

- Global logs are maintained for the system and include data describing the entire system workload activity. Job logs are provided for each job and describe job activity.

- Logging information is separated into two classes, coded information for display processes and binary information for compression purposes.

- Logs provided include:

  - Global logs
    System log ("system dayfile" in NOS/170)
    Account log ("account file/log" in NOS/170)
    Engineering log ("error log" in NOS/170)
    Statistic log

  - Job-local logs
    Job log ("job dayfile" in NOS/170)
    Job statistic log

Job classification

- Job class is the way of delegating priority.

- Job type is the way of determining the system capabilities available to the job. A transaction job, for example, may have different services than a batch job.

- Job mode is either interactive or batch.

Job naming

A job is identified by:

JOBNAME.USER_ID.SYSTEM_ID

JOBNAME is a user supplied field which must be given for batch jobs. It is assigned by the system for interactive jobs (terminal).

USER_ID is the combination of user name and family name which together form a unique user identification.

SYSTEM_ID is a unique system identifier provided by the system and may be used by the system operator for controlling jobs.

Recovery

Several levels of recovery from system failures are provided by NOS/VE and include:

- Recovery of jobs from the last system checkpoint. The

checkpoint may have been initiated by the operator, the system, or the on-line monitor for hardware or environmental reasons.

- Recovery of a single job from the last user checkpoint.

- Recovery of job input and output queues.

Error detection capabilities provided by CYBER 180 hardware increase the integrity of recovery operations.

2.2 PROGRAMS/TASKS

A program is a set of object modules and/or load modules organized to perform some specific function (e.g., compile COBOL statements).

A task is the execution of a program. Tasks are the only asynchronous execution element supported by NOS/VE. Tasks are protected from one another, can be dynamically created and destroyed and can communicate and synchronize with other tasks using system supplied requests.

The use of tasks as the basic NOS/VE execution element supports the transaction processing mode required of NOS/VE.

Important aspects of a program/task include:

- An object program is prepared for execution by the loader. Each object program may have its own list of object libraries which are used to resolve external references.

- An object module or load module is produced by a compiler and or linkage editor and consists of an instruction section, a binding section and a working storage section. The loader assigns sections to segments prior to execution. The instruction section is not modified and can be shared between tasks. Each task will have a private copy of the binding and working storage information.

- An object library contains multiple load modules and an entry point dictionary for those modules. A library is contained within a single protection level (RING). The protection attributes are part of every file catalog entry. Entry points available to levels with less privilege are referred to as gated entry points. Only library entry points may be gated.

2.0 CONCEPTS
2.2 PROGRAMS/TASKS

- Each object program will typically consist of user supplied object modules and operating system modules. Operating system modules will be protected from and will execute at a more privileged level than user modules.

- NOS/VE job local queues are used for communication between tasks. A short message or an entire segment may be passed between tasks.

The distinction between a program and the execution of a program is maintained for the following reasons:

- Multiple executions of the same program can be requested and the calling task must be able to identify each execution.

- Single execution of a program (serialization) can be enforced by the system.

- Tasks perform work on behalf of a calling task. The called task must be able to inherit priority, privilege and accounting attributes from its caller; i.e., a user task calls a system task.

2.3 FILES

Files are a logical container for data and/or a source or destination for data. NOS/VE supports permanent files (registered with the system in a catalog and saved for subsequent access) and temporary files (discarded at job termination). The file system provides a data path to all peripheral devices and provides a set of device-independent functions to application programs and end users.

Important aspects of a file include:

- All permanent files have a single owner. Only the owner is allowed to change the definition and identification or to delete a file. The owner also specifies and maintains the access control.

- Access control lists are the primary mechanism used to manage access to permanent files. The owner can generate an access control list which specifies the identification of users who can access a file and states how they may access it.

- Each permanent file is identified by a permanent file name. The name can be up to 31 characters in length and is unique

relative to the catalog in which the file is registered. Multiple versions of a permanent file may be registered under one permanent file name. Each version is called a cycle. A local file name (LFN) is a name used within a job to identify a file to be accessed.

- All files processed by a job are represented by a file descriptor that contains the attributes used to describe and manage the content and processing of files. The descriptor for a permanent file is retained for the life of a file. Use of the file descriptor allows all programs (e.g., EDITOR, COMPILER, command language, application) that process a file to interrogate the descriptor and determine file characteristics that affect their processing. This consolidates specification of attributes with the file management functions and eliminates the redundancy and complexity associated with having the user restate attributes to every program that processes the file.

- The organization of a file defines its logical structure. It is based on the way data is placed or located in the file by the Basic Access Methods. The basic file organizations provided are:

Sequential

Records are stored and retrieved in the order in which they are presented to the file system. A sequential file can be assigned to any one of the types of devices available to a user (disk, magnetic tape, terminal). Forward or backward positioning by number of records/tape marks/blocks/partitions is allowed. When a positioning function is not meaningful for a device (such as terminal), it is accepted and ignored. Delete and rewrite (same length) of the current record are allowed on disk files. Partial records are supported and one level of partitioning is available.

Space is allocated to a sequential file as the user transfers records to the file. The records are logically contiguous within the file. Disk space allocation and disk/tape volume switching occurs as necessary.

Byte-Addressable

Records are stored and retrieved based on the byte address of the start of the record. The file is viewed as containing continuous space for data (a string of bytes). Bytes within the file are numbered 1 to n. A

byte-addressable file can only be assigned to a mass storage device. It can be accessed randomly by byte address or sequentially. Forward or backward positioning by number of records/partitions is allowed.

Delete and rewrite (same length) of random records is allowed. Partial records are supported and one level of partitioning is available.

Space is allocated to a byte addressable file based on a comparison of the specified byte-address and the current end of information (EOI). If the byte address is greater than EOI, the file is automatically extended. The records need not be logically contiguous and therefore areas in which the data is undefined can develop.

Random and sequential access can be mixed within a file (positioned randomly followed by sequential record access).

- Access to files may be record oriented (via GET, PUT), physically oriented (via READ, WRITE) or segment oriented (via machine memory reference operations).

Record and Read/Write Access

This type of access is the primary user interface to files and requires usage of the file management functions such as GET, PUT, READ, WRITE to transfer data between a user memory area and a file. This access mechanism provides buffering, error handling, record management, and management of the structures of the various file organizations. Access is available to a variety of peripherals - tape, RMS, terminal, etc.

Segment Access

Segment access is available only for mass storage files and allows a file to be associated with a virtual memory segment and referenced with machine instructions. The user is responsible for interpreting and/or creating the content of the file. The movement of data between virtual memory and the external file is managed through the memory management functions.

2.4 SEGMENTS

Virtual memory is organized as a set of memory segments. NOS/VE manages both temporary and permanent segments.

Temporary segments contain temporary data such as working storage, heaps, etc. They are managed by explicit segment management requests and exist no longer than the life of the job in which they are created. Temporary segments allocated by the user are limited to Read, Write, Read-Write, and Execute protection.

File segments are used for referencing data in a temporary or permanent mass storage file. They are implicitly managed by the operating system as a result of an explicit file management or task management request (e.g., ATTACH a file, load a program).

Code segments are allocated by the loader and have Read and Execute access. A temporary code segment cannot be shared, whereas a file segment containing code may be shared.

All active segments are backed by allocated mass storage commensurate with current segment size. Current segment size is determined by the largest byte address referenced during the life of the segment. Segments expand and contract linearly; they do not contain unallocated empty spaces.

All active segments are assigned a segment number. The segment number is part of the Process Virtual Address (PVA) used to reference a byte in a segment. When segments are shared among tasks, it is likely that a different segment number will be assigned to each instance of the segment in the various task address spaces. The user has no control over assignment of segment numbers.

Permanent file segments may be shared among tasks of one or more jobs. Temporary segments may only be shared among tasks within a job. Read, Write and Execute ring brackets may be set differently for each task which has access to a shared segment. Read and Write protection may be specified differently for each task which has access to a shared segment. For a given segment, Write and Execute protection attributes are mutually exclusive.

2.5 ERROR HANDLING

During execution a procedure may encounter a set of error conditions which prohibit it from continuing in the normal manner. Examples are an attempt to divide by zero, or being unable to find a necessary file in the storage system. Circumstances that are abnormal for one procedure may be quite normal when encountered in a different procedure. If it is unable to continue, a procedure will want to notify its caller and others of its ancestors that an unusual occurrence has taken place.

The large number of error combinations require that detection and notification be done in a uniform manner throughout the system. Status and conditions represent two basic methods which are used for error detection and notification within NOS/VE.

2.5.1 STATUS

The general approach for handling errors is that the detector of an error shall return all relevant information about the error in a status variable. The status variable is a record that contains a status code, a detecting product identifier, a message identifier and optional additional message text. All major interfaces, both program (PASCAL) and command (SCL) must provide for a status variable.

2.5.2 CONDITIONS

Status codes enable a procedure to take action on an unusual occurrence after the procedure encountering the occurrence has returned. It is sometimes necessary for a procedure to gain control immediately upon encountering an unusual occurrence. Conditions are the mechanism which support this feature.

Conditions are divided into classes which include hardware, system, language and user. The condition mechanism is activated whenever one of these conditions is detected. The standard task environment will take default action on these occurrences if user processing is not selected.

2.5.3 PRINTED MESSAGES

The type of error reporting that most users first encounter is a message printed on their terminal or dayfile. In order to insure uniformity in message handling, all software components use a system message generator whose primary input is the status record.

2.6 SYSTEM EXPANSION

A base system that can grow and change over time is required. To allow for system expansion, a set of mechanisms are used that allow it to be extended.

1) PASCAL is the primary system implementation language. Its data typing facilities make it more extendable than other non-typed higher level languages and assembly language.

2) The flexibility of the CYBER 180 segmented address space allows the easy mixture of multiple independent units of code in the same addressing context. Code may be included within the same address space and still be protected via the ring and key/lock segment attributes. The segmentation addressing mechanism also allows the instruction portion to be shared among all users; that is only one physical copy of the instructions exists in a single mainframe regardless of how many users are currently accessing it. This combination of sharing and protection of instructions inside an addressing context allows a module to execute in its most convenient environment with negligible overhead.

3) The tasking facility of the operating system provides a synchronous or asynchronous component of a job. This may be utilized by any user job, by a application program running in the normal user environment or by the system itself as a mechanism for dynamically organizing work to be performed.

   A list of libraries to be utilized for satisfying external references may be specified at task initiation time. The libraries specified in this list may be generated by the user or may be system provided. The user has the capability to add to, delete from and order these library lists, thereby allowing replacement of a system provided module.

4) The command interface to the system is designed in such a way that new user commands (i.e., command procedures) are

syntacticly equivalent to system defined commands. This allows part of the interface to the system to be developed as command language procedures.

2.7 SECURITY/PROTECTION

A basic objective of NOS/VE is to provide efficient safe services to multiple users simultaneously and asynchronously. Levels of service to be provided range from complete isolation of users from each other to controlled sharing between cooperating users. In order to allow this range of service levels, the system has adopted a general access control strategy or security model which serves as the conceptual basis for the detailed implementation of all the access control mechanisms in the system.

The access control strategy is based on a conceptual access control matrix. Rows of the matrix represent all possible users of the system or, as they are known, in the access control model, subjects. Columns of the matrix represent all possible "things" that can be accessed in the system or, as they are known in the access control model, objects. Each element in the matrix is identified by a (subject, object) pair. Each element in the matrix contains the valid kinds of access or access rights that the subject has to the object. Consider an example:

	OBJECTS				
SUBJECTS	USER A	USER B	FILE C	FILE D	TAPE DRIVE E
USER A		ADMINISTRATOR	OWNER R,W		OWNER USE
USER B			R	OWNER R,E	

In the example, user A is the administrator of user B, the owner of file c and tape drive E. A can read and write C and use E. User B can read file C and owns and can read and execute file D. Every access every subject makes to every object is validated via the access control matrix. The access is permitted only if the corresponding access right is in the appropriate element in

the access control matrix.

Obviously, the operating system cannot maintain a physical matrix which is consulted on every access. A variety of features of the system architecture interact to implement the conceptual access control matrix. Major features of the NOS/VE include:

- Security level
  - Security level is an attribute of the system, users and system elements (e.g., files, devices).
  - The security level of the system is established by an authorized operator.
  - The owner of a system element specifies its security level.
  - A user is validated to access the system at specific security levels. The login process establishes a security level to be associated with a batch job or interactive session.
  - Security levels are always checked when validating a user's access to the system or system elements. The security level of the system and user must correspond to the security level of the system element on order to gain access. The user must also be permitted to access the system element.
- User identification and validation
  - a user must be known before gaining access to the system
  - the resources a user can use are a function of user controls, accounting controls, the current state of the system, and the security level of the system
  - an attribute of every user is the lowest ring number of execution execution. This controls functions available to a user.
  - modificaton to the user validation information may only be performed by the system, account and project administrators who control the user's installation
- File system
  - all files in the system are owned by a single user
  - access to permanent files is regulated by an access control list that is associated with each file. The file owner establishes the access control list. It contains the names (user identification) and access rights of all users permitted to access the file.
  - all files have one or more ring brackets associated with them which are used as qualifiers to file access
    - if a file is readable, then it possesses a read bracket which defines those rings in which it can be read
    - if a file is writable, then it possesses a write bracket

which defines those rings in which it can be written
- if a file is executable, then it possesses an execute bracket which defines those rings in which it may execute and a call bracket which defines those rings from which it may be called

The ring brackets associated with a file are specified by the owner of the file. However, the file system will not allow any user to specify any ring bracket of higher privilege than the ring in which the user is executing.

Segment management
- For a file accessed through the virtual memory mechanism, the file protection attributes maintained by the file system are used by segment management to build the segment descriptor table entries used by the CPU address translation logic when referencing the segment. The attributes the file system software have maintained are continuously enforced by the hardware when the file is being referenced.
- The loader accesses all object libraries through the segment level access facility of the file and memory management systems. It also uses segment management to create the transient segments that are used for the data areas of the executing program. The loader is responsible for creating these segments with the correct protection attributes to assure proper execution and protection of the program.

2.8 VIRTUAL ENVIRONMENT

Virtual Environment is defined as the shared use of a CYBER 180 mainframe by NOS/VE and NOS/170 or by NOS/VE and NOS-BE/170. It provides the following capabilities:

- A migration aid to assist user and application conversion to CYBER 180. Jobs and files can be transferred between the two environments.

- To provide CYBER 180 product features prior to developing the CYBER 180 counterparts. NOS/VE release 1 will support job entry and disposal, operator communication and networking by "front-ending" NOS/VE software. Communication and local batch facilities provided by CYBER 170 are used by NOS/VE.

Important aspects of the virtual environment are:

- The design base is a linked mainframe perspective. This

implies two distinct systems with well defined and controlled interfaces between them. Whenever possible, interfaces between functions are symmetric between 170 and 180 environments.

- Users and operators are aware of the two environments. From the users perspective, a job is either a CYBER 170 job or a CYBER 180 job, a file is either a CYBER 170 file or a CYBER 180 file, etc. Explicit CYBER 180 and CYBER 170 commands/requests are available to interface to the opposite state (e.g., route a file, submit a job, status a submitted job, get a copy of a file). Location is included as a parameter.

- Installations have options (post version 1) as to "front-end" functions that are performed by the CYBER 170 or CYBER 180 system. Examples include input/output spooling, operator control, network management, etc. Initially, NOS/VE requires some of these functions in CYBER 170 state due to development constraints.

- Memory, PP's, channels and peripheral devices are partitioned between the 170 and 180 operating systems. Dual access controllers may be split between states. Explicit system and/or operator functions are required to alter the respective configurations (e.g., operator action to reassign peripheral).

- Internal interfaces and mechanisms for communication between the states are under system control (requested via OS services).

- An interactive or remote batch terminal user can login to either the CYBER 170 or the CYBER 180 operating system.

- One MCU supports both states. Each environment maintains its usage and error log information for each system component that is either wholly owned or shared.

3.0 FUNCTIONS

3.1 JOB MANAGEMENT

3.1.1 GENERAL RESPONSIBILITY

The job management function is responsible for controlling the execution of jobs. These functions include scheduling, initiation, command statement interpretation and termination of jobs. Job management supports command and program interfaces which allow jobs to be created, submitted for execution, statused, and terminated.

3.1.2 GLOSSARY

System Command Language (SCL) - The language with which the external user communicates with the various components of the operating system.

Prolog - A set of commands that are processed automatically by the SCL interpreter on behalf of the user when logging in.

Epilog - A set of commands that are processed automatically by the SCL interpreter on behalf of the user when logging out.

SCL Interpreter - The set of system routines responsible for processing statements written in SCL.

Standard Input File - The file specified as input on the submit command. It is the source of commands for the SCL interpreter.

Alternate Input File - Any file other than the standard input file which is processed as input by the SCL interpreter.

Current Input File - The file currently being processed as input by the SCL interpreter.

Resource, Non-preemptible - A resource which cannot be given up to other jobs. Tape drives and private permanent files are examples of non-preemptible resources.

Resource, preemptible - A resource which can be given up to other jobs. Memory, system table space, processors and channels represent preemptible resources.

3.1.3 DESIGN OBJECTIVES

- Recovery of executing jobs, queued jobs and queued job output is a major design parameter in the definition of job and system structures.

- NOS/VE supports an execution environment which includes transaction, interactive, and batch jobs.

- The design will provide a reasonable compromise in terms of time and space required to schedule and allocate resources in a timesharing, batch and transaction environment which contains many jobs and resources with conflicting requirements. The following guidelines are used in scheduler design:

  - Allow tasks to request or release a number of resources with a single operation for both efficiency and deadlock prevention.
  - Where appropriate, allow a resource to be accessed in either sharable or non-sharable mode.
  - Allocate resources on a priority basis.
  - Make minimal restrictions on the user.
  - Avoid idle resources.
  - Allow a task to continue processing after a resource request is denied, if the task can do so.

- Job management allows the user and/or system a way to alter the environment of a job. This includes commands available, files available, default values for scheduling, run time libraries available and limits on various resources.

- Complete job status information relative to submitted and executing jobs must be available for users and operators.

3.1.4 ASSUMPTIONS AND CONSTRAINTS

- The batch job deck structure of NOS/170 is used for NOS/VE. The degree of compatibility is influenced by:
  - A NOS/VE objective is to make the batch and interactive interface compatible. The NOS/170 structure is batch oriented and in some cases may compromise interactive interfaces.
  - Support of the multipart file on input causes some special case handling by the system.
  - NOS/170 users are making more use of procedure files and interactive access. By the mid 1980's the importance of the original deck structures will have diminished.
  - NOS/VE supports one level of partitioning within the input file.

3.1.5 DESIGN APPROACH

A job represents a basic unit of work which can be directed to a specific mainframe within a configuration. Each job runs on behalf of a single user. The user identity is authenticated at job initiation and is relied upon for the duration of the job.

All jobs are protected from one another. Jobs compete for resources on a priority basis. The system maintains a clear distinction between preemptive resources and non-preemptive resources. Real memory represents a key preemptive resource managed on a job basis. Paging, a system responsibility, is used for managing real memory within a job and in general replaces overlays as a memory management technique. Swapping is used for managing real memory across multiple jobs, allowing the system to have more jobs initiated than the memory can hold.

The scheduling functions are periodically activated allowing for reevaluation of resource usage and job priorities. All batch jobs are queued prior to entry into the execution queue and selected for initiation based on user supplied priority, class and mode. Provision is made to also execute an installation defined procedure as part of the initiation process.

The NOS/VE procedure capability is the mechanism by which users can provide an external interface oriented to special

functions they require. Consistent conventions for calling and defining command language procedures are defined. The syntax of calls to command language procedures is identical to that used for calling system procedures (command language procedures or pascal procedures), thereby eliminating barriers between the interface provided.

The initial characteristics of the job environment are cataloged within the system according to user name. Included in the user description is a set of commands that are interpreted before any user commands (excluding login) are processed. These commands are referenced as a prolog. This allows defaults, search lists, time limits, exception conditions, additional security checking, etc. to be uniquely established for each job in an automatic fashion. The name of an epilog procedure is also included for handling cleanup, file routing and other services at the time of job termination. Each user can supply the prolog and epilog procedures for tuning the environment of jobs which run under his user name.

Included in the job environment are a set of system established files (input, output, job log, ...), segments (task services code, job local tables, ...), variables (command status, rerun status, ...) which have unique names that the job can reference on system commands.

A NOS/VE batch job consists of an SCL partition followed by any number of data partitions.

Jobs defined for NOS/VE which originate from C170 will be preceded by a routing partition. This partition contains a single record which is a C170 job card.

The C170 job card must be of the following form:

JOBNAME,ST<ID>,IC<cset>.

where:

ID - specifies a logical ID of the destination mainframe

cset - identifies the character set in which the command partition is coded. Supported valued for cset are:

AS6 - NOS 6/12 ASCII
AS8 - C170 8/12 ASCII
DIS - display code

Each origin system has its own default value for IC.

3.1.6 NOS/170 DIFFERENCES

System Command Language - The objective to produce an interface which is more consistent, more usable and less redundant than the NOS/170 interfaces has resulted in command language syntax incompatibilities. The improved interface will result in improved usability and reliability.

3.2 FILE ROUTING

3.2.1 GENERAL RESPONSIBILITY

The file routing function is responsible for routing queued (spooled) files to the proper physical locations for disposal or execution. Routing may be directed implicitly by installation or system conventions or explicitly by user commands and parameters. Destination addresses may specify logical characteristics such as "ASCII line printer" or "terminal logged in by user ABC". Files may be routed among mainframes connected by network or other link mechanisms. The status of these various files may be displayed and certain of their characteristics may be altered by command.

3.2.2 GLOSSARY

- Batch device: a remote batch terminal or a set of local batch devices. A batch device is "logged in" by an operator who may submit input files and recieve output files on behalf of other users.
- Executing job: a job that is in some stage of executing user specified activities.
- Input file or input job: a queued file having execution as its next processing step.
- Output file: a queued file having disposition to an output device (printer, punch,...) as its next processing step.
- Queued file: a file residing on intermediate storage (normally system disk) after inputting from or before outputting to a terminating device such as a card reader or printer. While the file is on intermediate storage, it is known to the system, which selects these files for further processing (execution, printing,...) based upon some scheduling discipline such as first-in-first-out, priority, etc. Queued files are either input files or output files.

3.2.3 DESIGN OBJECTIVES

The file routing function will provide file and job routing, visibility and control capabilities.

- Routing
  - of input jobs to various connected mainframes in order for:
    - jobs to have access to needed resources (certain permanent files, certain CPUs, peripheral devices,...)
    - the system to allocate jobs to computing resources for load leveling purposes
  - of output files to various destinations or devices in order to:
    - make the files accessible by the routing user
    - make the files accessible by other users or members as desired by the routing user
    - cause the file to be disposed in a desired physical format
- Control
  - of input jobs in order to allow:
    - reconfiguration of multimainframe resources to various mainframes (e.g.divert job specified to execute on mainframe A to mainframe B)
    - alteration of the order in which jobs will be selected for execution
    - dropping or purging of jobs
  - of executing jobs to allow:
    - dropping or purging of jobs
    - alteration of the service level which jobs will get in regard to system resources and services
  - of output files to allow:
    - reconfiguration of output devices (e.g.divert output nominally destined to terminal A to terminal B)
    - alteration of the order in which files will be disposed
- Visibility
  - of the current status of input jobs, executing jobs, and output files in order to permit informed control over jobs and files

3.2.4 ASSUMPTIONS AND CONSTRAINTS

- A batch device operator or interactive terminal operator who submits batch jobs must have full visibility and limited control of all input and executing jobs originating from the terminal and all output files destined for the terminal or device.

- The job owner (user specified on the job statement) must have full visibility of all owned jobs and files regardless of how they were submitted or routed.

- The system operator must have full visibility and limited control of all jobs.

- A "file label" is present on output files and is available to agencies that dispose output. This label contains information that is relatively independent of any specific instance of file disposition. This applies to things like internal character set.

- The batch capabilities of the system include an ability to define a set of devices (e.g.2 card readers, 3 printers) that can be "logged in" under one member ID without violating the assumption that only one person uses a given user or member ID.

- Both commands and procedure calls must be provided for file routing, control and visibility functions. Commands and procedures must have equivalent capabilities.

3.2.5 DESIGN APPROACH

Input, executing and output files each have a set of attributes which determine their routing and disposition. Some attributes are specified at the time the file is routed, and others are specified by other mechanisms. All may be included in status information to keep users and operators apprised of the progress and status of the jobs and files. Certain attributes may also be altered after the job or file has been routed. The following subsections describe the routing-related attributes and the conditions that must be met for their status to be displayed and for control over them to be exerted.

3.2.5.1 Input File Attributes

Existence: whether the file is known to the system. Ability to control existence is the ability to drop or purge the file.

Name: the name by which the file is known. This name is displayed in status information and is used in control commands to refer to a particular job. The file is given a name by a parameter to the ROUTE command or procedure - or if none is specified, the name from the job statement is used.

Mainframe to execute: the identification of the mainframe that is to execute the input job. This may be specified in the ROUTE command or procedure call and if none is specified, then it is taken from a parameter on the job statement. If none is specified on the job statement, an installation parameter is used (normally the current host mainframe).

Default output file destination: the default destination of all output files generated by the job. This may be specified by a parameter to the ROUTE command or procedure. Individual output file destinations may be specified by individual ROUTE commands or procedure calls within the job.

Originating family and member: the identification of the batch or interactive operator by whom the file was submitted or the identification of the member currently in force at the time a file was submitted from a job.

Originating mainframe: the mainframe to which the submitting device is connected or the mainframe in which the submitting job is executing.

Protect or not: whether the job should be recovered after system re-initialization. This is normally used only by system components - for example, "spot"s.

Dependency information: specifies job dependency information regarding the conditions necessary for this job to execute.

Priority: the relative order in which input files will be selected for execution may be implied from this attribute. This may be specified by a parameter to the ROUTE command or procedure or if none is specified, a parameter on the job statement is used. If no job statement parameter appears, a member validation parameter is used. Specified parameters are subject to system access limits placed upon the owner member of the job.

3.2.5.2 Executing Job Attributes

Existence: whether the job is known to the system. Ability to control existence is the ability to drop or purge the job.

Name: the name by which the job is known. This name is available in status information and is used in control commands to refer to a particular job. An executing job's name is the same as that given to it when it was an input file.

Protect or not: see input file attribute of same name.

Priority: the relative priority a job is given for system resources and service.

State: whether the job is currently executing or waiting for some event. If the job is waiting, the reason for the wait is also available:

- access to permanent file (file is interlocked)
- pack mount
- tape mount
- operator response
- ...

Executing mainframe: the identification of the mainframe that is executing the job.

SRUs accumulated: the number of SRUs expended so far in the execution of the job.

Position: an indication of how far the job has advanced in its execution. For example, last dayfile message, last command, some programmable message,...

3.2.5.3 Output File Attributes

Existence: whether the file is known to the system. Ability to control existence is the ability to drop or purge the file.

Name: the name by which the file is known. This name is available in status information and is used in control commands to refer to a particular job. The file name may be specified by a parameter to the ROUTE command or procedure. If this parameter is not given or if automatic end-of-job output file routing occurs, the file name is that of the job that generated or routed the output.

3.0 FUNCTIONS
3.2.5.3 Output File Attributes

Originating family and member: see inout file attribute of the same name.

Originating mainframe: see input file attribute of the same name.

Destination mainframe: the identification of the mainframe that is connected to the device to which the file is to be disposed. This may be specified by a parameter to the ROUTE command or procedure. If this parameter does not appear, the mainframe that originated the job is used.

Destination family and user: the user who, when logged in at a batch device or station, is to recieve the output. This attribute may be specified by a parameter to the ROUTE command or procedure and if none is specified, the family and user of the job that generated the output or is doing the routing is used. The ability to restrict which users will be allowed to dispose output to a terminal is given to terminal users. Therefore, the routed file may not be permitted to be disposed to the destination terminal.

Destination type: the type of destination of the output file. This may be specified by a parameter to the ROUTE command or procedure. If the parameter is not given and the local file name is one of those listed below, a default type is selected based upon the file name. Destination type may be:

- printer
- card punch
- installation-defined types

The special file names used for default type selection are:

- output selects "printer"
- punch selects "card punch" (and external characteristic "coded ASCII")
- punchb selects "card punch" (and external characteristic "punch binary")
- installation-defined file names

External characteristics: the form in which the output file is to be disposed. External characteristics may be specified by a parameter to the ROUTE command or procedure and if not specified, defaults are selected based upon destination type or special local file name. External characteristics may be:

- 96 character ASCII (type "printer")
- 64 character ASCII subset (type "printer")
- punch coded ASCII (type "punch" or local file name PUNCH)
- punch binary (type "punch" or local file name PUNCHB)
- forms code (types "printer", "punch", installation-defined) (may be combined with another characteristic)

Type "printer" default is "96 character ASCII"; "punch" default is "punch coded ASCII" unless local file name is PUNCHB; "installation-defined" default is installation-defined.

Priority: the relative order in which output files will be selected for disposition may be implied from this attribute. Priority may be specified by a parameter to the ROUTE command or procedure and if not given, an installation-defined priority default is assumed. This may be specified on ROUTE only if the job or terminal doing the ROUTE is owned by the same user as the destination user.

Address: location information to be used to get physical output to a particular physical location (e.g.mail drop). Address information is printed, punched, etc. in a fixed location in the output listing, deck, etc.

Repeat: the number of copies of the file to be produced at the destination. This may be specified by a parameter to the ROUTE command or procedure and if not specified, is assumed to be 1.

Size: the size of the output file.

3.2.5.4 Routing, Visibility and Control Matrix

Type	Route	Other	Status 1	Control
Input				
Existence	-	-	ju,ou	ou,so
Name	X	X 2	ju,ou	
MF to execute	X	X 3	ju,ou	so
Default output destination	X		ju,ou	ou,so
Originating family and member			ju,ou	
Originating MF			ju,ou	
Protect or not	X		ju,ou	
Dependency information		X	ju,ou	
Priority	X	X 4	ju,ou	so
Executing				
Existence	-	-	ju,ou	ou,so
Name	-	-	ju,ou	
Protect or not	-	-	ju,ou	
Priority	-	-	ju,ou	so
State (executing,waiting)	-	-	ju,ou	so
Executing MF	-	-	ju,ou	
SRUs accumulated	-	-	ju,ou	
Position	-	-	ju,ou	
Output				
Existence	-	-	ju,du	du,so
Name	X	X 5	ju,du	
Originating family and member	-	-	ju,du	
Originating MF	-	-	ju,du	
Destination MF	X	X 6	ju,du	du,so
Destination type	X	X 8	ju,du	
External characteristics	X	X 9	ju,du	
Priority	X		ju,du	du
Address	X	X10	ju,du	
Repeat	X		ju,du	du
Size	-	-	ju,du	

Symbols Used

Route: the attribute is specifyable by a route command or procedure parameter or a default is chosen by route.

Other: the attribute is specifyable or defaults are chosen by mechanisms other than the route command or procedure.

Status: the status of the attribute is available to these users.

Control: the attribute may be altered by these users.

X: available via mechanism described in column header.

-: does not apply.

ju: the user who is the owner of the job or file. This would be the person named in the job statement of a job.

ou: originating user of the job or file. This is the user that was logged in at a batch device through which the job was submitted or the owner user of the job that routed the file.

du: destination user. The user to which the file is to be routed.

so: system operator.

Footnotes

1. System operator may get all status.
2. Job name from job statement is default.
3. Parameter on job statement is default.
4. Parameter on job statement is first default, default information from member validation is second.
5. Name of the job doing the routing is the default.
6. Mainframe of the job doing the routing is the default.
7. Family & member of the job doing the routing is the default.
8. Special file names select the default type.
9. File type and special file names select the default.
10. Owner user validation information is used as the default.

3.2.6 NOS/170 DIFFERENCES

- The "device ID" concept is replaced by allowing local batch device sets that can be treated like remote batch terminals. That is, routing to a specified user or member "terminal".

- Specific routing to 512, 580-12, 580-16 and 580-16 is removed. Similar capabilities could be presented using batch device sets as described above. Disposition to particular devices in this sense is normally decided by the disposing agency (e.g.batch facility) based upon considerations such as priority, file size, etc.

3.0 FUNCTIONS
3.2.6 NOS/170 DIFFERENCES

- Certain "external characteristics" have been removed:
  - 48 character ASCII (print)
  - 48 character CDC scientific (print)
  - 64 character CDC scientific (print)
  - 026 code (punch coded)
  - system binary (punch binary)

- Internal characteristics of character set and 580 spacing code are removed from routing commands and procedures. Information in the file "label" is used instead.

- The information available about a file or job is expanded from that available via the STATUS or ENQUIRE commands.

- A job's originating user ability to control certain aspects of the file or job attributes is added.

- Added the ability to route to a specified mainframe.

- Added the ability to specify address information for output files.

- Added the ability to specify default output attributes at the time a file is routed to input.

- Added the ability to route output to any user - with access control capability at the destination terminal.

3.3 PROGRAM MANAGEMENT

3.3.1 GENERAL RESPONSIBILITY

The program management function provides for the initiation, monitoring and termination of program execution. This includes the command processing for program execution and library generation and the program interface processing for program execution, program communication, condition processing and general program service requests.

3.3.2 GLOSSARY

Signal - A short message used for inter-job/inter-task communication.

Task - An instance of execution of a program. A task has an associated exchange package and segment table defining its address space. Tasks are the basic NOS/VE execution element.

Task Services - A selection of protected operating system procedures in the address space of a task. To be supplied

3.3.3 DESIGN OBJECTIVES

The design objectives of program management are to provide the following capabilities in a manner consistent with the objectives and constraints of the NOS/VE system architecture.

- Allow execution of object programs from the command level.
- Allow synchronous and asynchronous object program execution from the program level.
- Allow communication between asynchronously executing tasks within a job.
- Allow communication between jobs.
- Contribute to the provision of the CYBER 180 segmented virtual

memory.

- Provide mechanisms to organize programs for efficient execution in a virtual memory environment.
- Allow object program binding to be performed at a variety of times in order to optimize for flexibility or performance.
- Provide sufficient services to allow program level access of the operating system to determine required information about the current environment (e.g., time and date).

3.3.4 ASSUMPTION AND CONSTRAINTS

- Commands, command procedure calls and command level program execution should be syntacticly identical.
- Program management command interface should be consistent with the externalization of the rest of the system.
- The CYBER 180 segmented virtual memory will be supported effectively in the first release of the system.

3.3.5 DESIGN APPROACH

3.3.5.1 Program Execution

Every job consists of one or more executable elements called tasks. Every task is viewed as an instance of execution of an object program. That is an object program is the static definition of an executable element including a name, a list of files containing object modules and a list of libraries from which external references are satisfied. The execution of an object program is known as a task.

The user interfacing to the system at the command level sees a single program description which may be explictly modified to control program loading and execution but through the use of defaults typically will not have to be. The command user is unaware of the concept of task and only dimly aware of the concept of programs. As the system evolves it may be desirable to make these concepts more visible at the command level.

At the program level, the user is aware of both tasks and

programs and should percieve tasking as a facility for organizing large applications and effecting asynchronism in a job.

3.3.5.2 Program Communication and Conditions

There are two levels of program communication: within a job and between jobs. Differences in the mechanisms available at the two levels are caused by the fact that the job is the fundamental unit of access control. Within a job less access checking takes place since everything within the job has already been verified for access by the user on whose behalf the job is running. Between jobs, of course, no such assumptions are possible.

Within a job, the mechanism for communication between tasks is a local queue. Local queues have names that are defined within the job and access to them is limited only by ring level. A segment offset and either a short string of bytes, or a pointer, or a descriptor(s) of entire segment(s) may be sent to and received from local queues. In either case, only a small amount of information is actually moved allowing the queuing operation to be relatively efficient.

Between jobs, two communication mechanisms are provided. The first is a signal which is a short string of data that is directed to a specific task. The second is a low level lock that is externalized as a semaphore or a global queue. If two jobs require the passing or sharing of large amounts of data, they must use segment level access to the same permanent file and use signals for control information. This technique allows the access control mechanism of the file system to control the validity of communication.

The condition processing facilities of the system allow any procedure in a task to establish its own procedures which will receive control when the condition arises. Examples of conditions for which handlers can be established are arithmetic overflow, divide fault and task termination.

3.3.5.3 Program Services

Program services provide tasks with information about the execution environment that is only known at execution time, for example time of day, date and current accounting statistics.

3.3.5.4 Program Management Command Processing

Program management commands are responsible for providing program execution and object library generation services at the command level. The general design direction for the program execution commands is to provide a straight forward unelaborate means for executing programs. The general design direction for object library generation commands is to provide the variety of packaging, binding, renaming and other facilities that in combination with the program execution facilities, provide the application, product set and end users with adequate support. The object library generation facilities should do as much processing as possible prior to load time in order to increase load time efficiency.

3.3.6 NOS/170 DIFFERENCES

Similarities and contrasts between NOS/VE and NOS/170 in the area of program management are as follows:

1. The notion of system commands, command language procedures and object program calls being syntacticly equivalent facilitating changing implementation of a particular command will be carried forward.

2. The CYBER loader provides mechanisms to support three basic functions: program loading, program binding and program structuring. These three functions will be performed by different mechanisms in order to support the CYBER 180 virtual memory architecture and the system command interface style. The CYBER 180 loader will be primarily responsible for loading. The object library generator will be primarily responsible for binding and structuring.

3.4 PHYSICAL INPUT/OUTPUT MANAGEMENT

3.4.1 GENERAL RESPONSIBILITY

The Physical I/O function is responsible for all phases of the actual data flow between the central processor and external devices. Physical I/O accepts requests from the logical I/O function (buffer manager) and other operating system components. Among the services provided by physical I/O are the following:

- Translation of logically oriented requests (file identifier, file byte address) to physically oriented requests (unit, cylinder, track, sector).
- Assignment of a device to a file.
- Space allocation on a device.
- Device I/O request queue management.
- Device driver (disk, tape, local unit record, communications, memory link, channel link).

3.4.2 GLOSSARY

Allocation Unit - The number of DAU's allocated to a file on a specific RMS device per allocation request. The allocation unit can be specified by the user on a REQUEST statement as the minimum (A0-1 DAU) or as power of two multiles of the minimum (A1-2 DAU's, A2-4 DAU's, A3-8 DAU's, ACY-1 cylinder). A default will be selected if the user does not specify the allocation unit. The specification can not be changed after the file is opened.

Block - A container for contiguous bytes of data. A block is represented on magnetic tape by the data contained between two physical interrecord gaps. A block is represented by the data area defined by a block control word for files with user specified blocking when the data appears in a memory buffer or on mass storage. A block is represented by a fixed amount of data for files with system default blocking.

Block Control Word (BCW) - Control information which precedes each block on a file with user specified blocking. The information includes the current block type, current block length, previous block length and error status.

Device Allocation Map (DAM) - The device allocation map contains a header and a bit map for the device. Each bit in the map corresponds to the device allocation unit. A separate DAM exists for each device.

Device Allocation Unit (DAU) - The basic unit by which a disc unit space is divided. The device allocation unit is determined by the device characteristics.

Device Allocation Table (DAT) - The DAT is a device resident table describing linkage and allocation information for files assigned to a particular device. A separate DAT exists for each mass storage device. The DAT is periodically updated to provide recovery.

Device Label - Information written on a device which identifies the device with a VSN and also contains other pertinent data concerning the device.

Driver - That code within the PPU which controls the transfer of data between CPU memory and a device.

Driver Queue Table (DQT) - There is a DQT for each driver. The DQT is the basic table used by the driver in performing its functions. The DQT links to the UQT table. During an I/O operaton, the DQT links to the IORP which specifies the I/O operation. This link is set by the queue manager when the request is selected.

File Allocation Table (FAT) - The FAT is a linked table which contains a FAT header and a sub-file header and body for each device on which a file is recorded. The device allocator uses fields in the sub-file header and updates allocation indexes in the body. The FAT is described in detail in the system I/O component specification.

FAT Header - The first section of a FAT which contains general information about the file.

Sub-file FAT - A section of the FAT after the FAT Header which contains allocation unit information about one device.

Sub-file FAT Heade- - The first section of the sub-file FAT which

contains information about this part of the file and the device upon which it resides.

Sub-file FAT Body - The remainder of the FAT sub-file following the sub-file FAT Header which contains the disk allocation information for this portion of the file.

FAT List Element - A contiguous portion of the linked list which holds a FAT. Any portion of the FAT except the FAT Header and sub-file Fat Header may span FAT List Elements.

Sub-file - A file contains one or more sub-files. A new sub-file begins whenever a different device is used.

Full Track - A recording format where contiguous physical sectors are allocated to a file.

Full-Track Data Access (Parallel FMD) - The disk driver operates on a full-track read and write basis by utilizing pairs of PPU's on an alternate sector access basis.

I/O Request Package (IORP) - A table used to hold the addresses of buffers in use by a file. Also used as a parameter list when making Queue Manager requests. Also used for control of overall I/O activity on the file.

Minimum Addressable Unit (MAU) - The quantum of data that can be accessed by a driver. The default MAU is 256 central memory words which is assumed to be a typical CYBER 180 page size. Accessing to RMS devices are always processed in terms of MAU's. The effective MAU for a file can be specified as a multiple of 256 central memory words.

Note: If the central memory buffer is less than the MAU, then the excess data may be skipped on a read and padded on a write.

Seek Overlap with Data Access - The positioning of a unit is carried out as a simultaneous overlapping action with data access of another unit.

System Default Blocking - Blocks are fixed in size. The size is not under the user's control but is specified by the operating system. The block is never preceded by a block control word.

Transfer Unit - The maximum amount of data transferred between

central memory "system" buffers and an RMS device per physical I/O request. The transfer unit can be specified by the user on a REQUEST statement as the minimum (T-1 MAU) or as power of two multiples of the DAU (T0-1 DAU, T1-2 DAU"s, T2-4 DAU"s) up to the allocation unit for the file. A default equal to the allocation unit will be chosen if the user does not specify the transfer unit. The specification of transfer unit can not be changed after the file is opened. Typically, the number of buffers in an IORP reflects the transfer unit size.

Unit Queue Table (UQT) - The unit queue table is the basic table used by the queue manager in supervising the processing of IORP"s. There is an UQT table for each disk subsystem. There is an entry in the UQT for each unit in the system. A header provides parameters required for the queue manager to select a data path to initiate requests.

UQT entries are used to contain:

- A link to the IORP queue for each unit.
- Parameters to determine the mode of IORP selection.

User Specified Blocking - Blocks are variable in size. The size can be specified by the user. The block is preceded by a block control word when it appears in a memory buffer or on mass storage.

3.4.3 DESIGN OBJECTIVES

- Support for reliability/recovery has highest priority
  - each mass storage volume must be self describing (label, allocation information, device characteristics (PF device, queue device, etc.)) flaw information, linkage of files residing on the volume
  - ability to recover files on mass storage after system failure without depending on memory (includes open temporary files)
  - ability to recover files on each mass storage volume after system failure based on information recorded on that volume. Information must include enough to reconstruct

tables to access the file (includes open temporary files)

- Support concurrent maintenance activities.
- Support preallocation of space including at a specific location within a mass storage volume.
- Support sharing of mass storage drives and files between mainframes.
- Support sharing of magnetic tape drives between mainframes.
- Support automatic mass storage device overflow.
- Support utilization of devices at their maximum rated speeds.
- Support flexible space allocation on RMS.
  - ability to allocate "reasonable" size areas of a mass storage volume to files with "reasonable" size central memory tables.

3.4.4 ASSUMPTIONS AND CONSTRAINTS

- Mass storage sector sizes are fixed per hardware type but may vary from one hardware type to another.
- The four head parallel FMD structure is such that a single PPU can not transfer sector data between central memory and PPU memory in the passing period of a sector gap.
- 180's can share RMS at the controller or drive level and do not require partitioning of the drives. Sharing between mainframes is coordinated at the controller and drive level.
- 180 can share RMS with 170 at the controller level and requires partitioning of the drives.
- 180's can share magnetic tapes at the controller level and requires partitioning of the drives.

3.4.5 DESIGN APPROACH

3.4.5.1 Separation of CP and PP Functions

The PPU is utilized only for device drivers and restricted to performing the operations required to function/status device controllers and transfer data between central memory and the controller. PPU I/O request parameters are in terms of physical device addresses (unit, track, etc.), physical central memory buffer addresses (real memory addresses) and function to be performed.

The CPU is utilized to translate logical parameters associated with an I/O request (file identifier, file position) to physical information. The CPU creates a physically oriented PPU I/O request packet and enters it into a queue associated with the appropriate physical device. The CPU selects PPU I/O requests to be executed and submits them to the appropriate driver. Selection of requests is optimized based on the characteristics and use of the equipment.

3.4.5.2 Support of Four Head Parallel FMD

The four head parallel FMD is full tracked in order to achieve the maximum transfer rate. This requires a two PPU driver since a single PPU can not transfer a sector between central memory and PPU memory in the passing period of a sector gap. The driver PPU's will process alternate sectors. Options to be considered include the following:

- single PPU driver for controllers without four head parallel FMD
- multi controller access from a single driver

3.4.5.3 Basic Physical I/O Support

Physical I/O is utilized by other operating system functions (pager, logical I/O) to manage the transmission of data between central memory and devices. The basic support provided by physical I/O includes the following:

- Mass storage allocation - Allocation is accomplished by finding the correct area on the device and reserving the space requested.
- I/O request queueing - Requests for I/O to the Physical I/O function are queued and scheduled based on the characteristics

and use of the equipment.

3.4.5.4 Residence of Functions

Physical I/O is partitioned into functions which reside in the task address space and those which reside in the monitor. The functions performed in the monitor are request dequeueing and response enqueueing. Functions performed in the task include translation from logical I/O request to physical I/O request, determining the path to the corresponding device, enqueueing requests and dequeueing I/O responses. Due to the residence of page fault processing in the monitor and its dependence upon physical I/O, certain functions normally performed in task services are also performed by the monitor in support of paging. These functions include disk allocation, request translation and request enqueueing.

Physical I/O mass storage functions are part of the monitor address space for the following reasons:

- Optimization of the path for page fault I/O requests.
- Optimization of the path for page allocation/deallocation requests.
- Less overhead in processing I/O requests from a job
  - One system call is used to transfer from job to physical I/O.
  - Call/return used within physical I/O CPU components.
- Locking/unlocking of system buffers is performed at the monitor level in order to minimize the length of time a system buffer is locked in real memory.

3.4.6 NOS/170 DIFFERENCES

File Structuring (partitions) is implemented at the logical I/O level (records) rather than at the physical I/O level (short PRU's).

Device drivers are to be considered as firmware.

Device drivers are statically assigned to PPU's rather than dynamically.

3.0 FUNCTIONS
3.4.6 NOS/170 DIFFERENCES

Device drivers do not have access to logical I/O table structures such as FET's, etc.

3.5 LOGICAL INPUT/OUTPUT MANAGER

3.5.1 GENERAL RESPONSIBILITY

The Logical Input/Output function can be utilized to remove the burden of physical I/O from the user. Logical I/O translates the user's logical I/O requests into requests to the Physical I/O function. Record oriented (record level access) input operations involve the transmission of data in physical format to Logical I/O buffers where they are translated and delivered in logical record format to the user. Record oriented (record level access) output operations involve the transmission of user logical records into Logical I/O buffers where they are translated into physical format and delivered to the Physical I/O function. Nonrecord oriented (read/write level access) input operations involve the transmission of data in physical format to the users buffer. It is the user's responsibility to interpret any logical record structure within the physical data. Nonrecord oriented (read/write level access) output operations involve the transmission of data in physical format from the users buffer. It is the user's responsibility to create any logical record structure within the physical data.

Specific services provided by Logical I/O include the following:

- maintains a description of file characteristics
- activate/deactivate file processing via OPEN/CLOSE
- access to a logical (record oriented) structure within a file via GET/PUT
- access to a physical (non-record oriented) structure within a file via READ/WRITE
- buffering of record oriented file access
- file label processing, error processing, etc.
- allows file access via explicit procedure call (GET/PUT/READ/WRITE) or by associating a virtual memory segment with a file and utilizing CPU memory access instructions to reference the data (segment access).

3.5.2 GLOSSARY

Advanced Access Methods (AAM) - Procedures which allow access to files via organizations such as indexed sequential.

Basic Access Methods (BAM) - Procedures which allow access to files via relatively simple organizations such as sequential access to the "next" record and random access to a record based on its address.

Blocking - The process of creating a block from part of a record or a series of records.

Collector Mode - Multiple data records coming from a terminal connected to a file will be entered into a system buffer before the requesting task is resumed to process the data. Multiple data records going to a terminal connected to a file will be entered into a system buffer before any record is sent to the terminal. Noncollector mode implies that the requesting task is resumed to process input data records one at a time and that each output data record is sent to the terminal as it is created by the user.

File - A file is a logically connected set of information. It is the largest collection of information which may be addressed by that file name. All data is stored between the beginning-of-information (BOI) and the end-of-information (EOI). Label groups are not considered to be part of file data in the general case.

File Description Table (FDT) - Contains file attributes and linkage necessary to access the file when it is active. The FDT contains links to the FAT and IORP's associated with a file.

File Label - Contains (among other things) backup for information normally maintained in permanent file catalog.

Partition - A partition consists of one or more records. It is less than a file and greater than a record; but it may be identical to either or both. A partition begins with the first record after the end of the preceding partition; a partition is delimited by a special non-data record.

Read Ahead - The process of anticipating a user's requirements by

physicaly transferring large amounts of data into system buffers on each physical I/O request. The assumotion is that the user will make successive requests for records in a forward sequential direction.

Read/Write Level Access - User accesses data directly in a buffer owned and managed by the user. Data transfer between the device and central memory will utilize the user buffer. This mode allows the user to control buffer size, number allocated, etc.

Record - A record is a group of related pieces of information. A record or portion thereof is the smallest collection of information passed between Record Manager and the user. The user defines the structure and characteristics of records within a file by declaring a record format. The beginning and ending points of a record are implicit within each format.

Record Level Access - Data is transferred between user memory area and system buffers owned and managed by a buffer manager. Data transfer between the device and central memory will utilize system buffers. A buffer manager exists for each task.

Record Control Header - Control information which precedes each record on a file with W, S and D records. The information may include the current record type (complete/continuation), current record length, previous record length and flags to indicate end of partition, deleted record. (Partitioning, deleted records and, efficient backspacing are only found in W records.)

Segment Level Access - User accesses data by associating a virtual memory segment with a file and utilizing CPU memory reference instructions to access the data.

System Buffer - Buffer owned and managed by operating system as contrasted with a buffer owned and managed by user code. System buffers can not be accessed by user code.

User Buffer - Buffer owned and managed by the user as contrasted with a buffer owned and managed by system code. User buffers can be accessed by user code.

User File Description - A table provided by the user and passed

as a parameter to the file system via the OPEN call in order to specify file attributes. The file system uses information from within the user file description to build the system file description.

Write Behind - The process of retaining user records in system buffers until a large amount of data can be transferred to the device per physical I/O request. The assumption is that it is more efficient to make one device access than many accesses.

3.5.3 DESIGN OBJECTIVES

- Basic Access Methods should satisfy requirements of the operating system, user and Advanced Access Methods with a minimum of I/O interfaces.

- File system needs to provide an interface by which a user can manipulate a record structured file at a level lower than a record (for example - disk sector, tape block). Example users are advanced access methods.

- If the advanced access methods access a record structured file at a level lower than a record but wish to manipulate the records, then it should be possible to interface to the code in the basic record manager which processes that type of record. Basic Access Methods should provide the "hooks" used by Advanced Access Methods to process additional file declaration parameters, record types, etc.

- Allow all mass storage files to be either explicitly accessed via a procedure call interface or association of a virtual memory segment with a file.

- Provide a sequential file organization that includes device independence at the record level interface (includes ability to access a file that is "connected" to a terminal.

- Provide a file organization which allows random access to mass storage files and provides a foundation for operating system structures, advanced access methods, etc.

- Provide a file organization which facilitates interchange of data with CYBER 170.

- Provide a single system default record type for record oriented files including job files, print files, etc.

3.5.4 ASSUMPTIONS AND CONSTRAINTS

- The total logical I/O interface can be divided into Basic Access Methods (provided by OS and defined in this specification) and Advanced Access Methods (provided by Product Set, includes indexed sequential, etc.).

3.5.5 DESIGN APPROACH

3.5.5.1 Residence of Functions

Logical I/O functions are associated with the task and reside within the task address space. The reasons for this association include:

- Less overhead than if logical I/O is at a separate task. The user interface to the record manager will be via CALL/RETURN. The record manager interface to the buffer manager will also be via CALL/RETURN.
- The linkage to the logical I/O functions can be tailored to the needs of the task.
- Information about files that are local to a particular user is protected from other users.

The Logical I/O functions are basically split into two categories; those dealing with the interface to the user (record manager, file manager) and those dealing with the interface to physical I/O (buffer manager).

- The record manager and file manager execute in a more privileged ring than the user in order to protect Logical I/O code, I/O request packets, system file description tables and system buffers from destruction or improper execution.
- The buffer manager executes in a more privileged ring than the record manager and file manager in order to maximize the protection given to the code that submits physical I/O requests.
- All tables used by Logical I/O to process file activity link through a system File Description Table maintained by Logical I/O.

- The file descriptors, I/O request packets and system buffers manipulated by Logical I/O are inaccessible to the user. Only the appropriate procedure entry points will be accessible to the user.

3.5.5.2 Major User Requirements/Options

The user must supply a user file description for each file used by a task.

The user may specify the file residence by supplying a REQUEST command/procedure call.

The user may supply information for use in file label checking/creation by a LABEL command or user label description.

The user may request that the file system provide whatever information it has retained about the file by using a GET_FILE_ATTRIBUTES call.

The user may parameterize the file description by supplying a FILE command.

The user must issue an OPEN call within each task for a file prior to accessing it.

The user may access/create labels on the file by using GETL, PUTL calls.

The user may transfer data to or from a file using the various versions of GET, PUT, READ and WRITE calls.

The user may position a file using the REWIND, SEEK or SKIP calls.

The option of "wait for completion" or "no wait" is available with READ/WRITE I/O requests.

The user may monitor progress of I/O requests submitted with "no wait" by using a CHECK_BUFFER call.

The user may submit multiple concurrent READ/WRITE requests.

The user may delete logical records by using DELETE calls.

The user may replace logical records by using REPLACE calls.

The user may introduce a file partition delimiter into his

file by an WRITE_END_PARTITION call.

The user may access/modify information in the File Description Table by use of the FETCH and STORE calls.

The user must terminate processing of a file within each task by using a CLOSE call. However, files not closed by a user within a task will be automatically closed by task termination.

3.5.6 NOS/170 DIFFERENCES

- Advanced Access Methods

  Indexed sequential, multiple indexed, direct access, and actual key file organizations are not provided as part of basic access methods.

- File Structuring at the physical level (short PRU, level numbers, etc.)

  One level of file structuring (partitions) is provided at the record access level (GET, PUT) rather than at the physical level (short PRU's). The purpose is to split logical file structure from physical recording technique. This facilitates implementation of a variety of sector sizes. The user interface is independent of a particular device recording technique (such as 3864 bit sectors on 844-4X versus 16448 bit sectors on parallel FMD).

- Circular Buffering

  Linear buffers are used in the low level (READ, WRITE) interface rather than circular buffers. The purpose is to simplify the interface to PPU drivers and to improve the ability to recover/reissue an I/O request when an error occurs.

- File organization is specified on the user file declaration and is removed from data access requests to reduce redundant specification of parameters.

- Additional block (E, K, I) and record types (T, S, R, non-ANSI D) have been removed from basic access methods to simplify the user interface.

3.6 MEMORY LINK

3.6.1 GENERAL RESPONSIBILITY

The Memory Link provides a general communication capability for linking the C170 (NOS/170 and NOS/BE) and the NOS/VE system. It will support communication between the C170 and the C180 systems for the following functions:

- Transfer of queue files and permanent files between systems.
- Allow C170 interactive terminals to communicate with NOS/VE jobs.
- Allow general message communication between a NOS/VE job and a C170 job.

3.6.2 GLOSSARY

CALL180 Instruction - The C170 017 hardware instruction which allows a C170 program to "trap" into C180 state. The C180 goes back to the C170 through the C180 RETURN instruction.

Memory Link Interface (MLI) - A set of program interfaces which allow a user or system application program to communicate with one or moe other user or system application programs. Both a C170 and a C180 version of these interfaces will be provided.

3.6.3 DESIGN OBJECTIVES

- Provide a flexible interface which can be used by both C170 and C180 sides for all Virtual Environment communication - file transfer, interactive and message communication links.
- Provide a symmetrical interface for both C170 and C180 sides (i.e., provide the same set of program interfaces (subroutines on C170) for both C170 and C180).
- Provide flexible interfaces which can also be used for C180 - C180 job to job communication.

- Place no additional C170 or C180 constraints on C170 job support or storage move.

- Place no additional C170 or C180 constraints on C180 job swapout or paging.

- Prevent access to the MLI by unauthorized C170 and C180 users.

- Prevent MLI from being used by C170 or C180 jobs to penetrate C170 or C180 system security.

- Minimize C170 performance degradation by keeping the time spent in the C170 "trap handler" (C180 code activated by the CALL180 instruction) to a minimum (e.g., all code and data referenced by the C170 "trap handler" must be wired down in order to avoid page faults).

- Allow all C180 MLI code to execute in Task Service of the calling task.

- The amount of "wired down" real memory used for MLI tables, buffers and code must not cause a significant performance degradation for C170 and C180.

- The Memory Link must not be a bottleneck for any type of dual state communication - file transfers, interactive or message communication.

- Provide record manager compatible program interfaces for user level system programs which allow data to be transferred to/from the linked C170 system for purpose of transferring queue files and permanent files. The record manager compatible program interfaces can be provided by mapping record manager calls into MLI interface calls.

3.6.4 ASSUMPTIONS AND CONSTRAINTS

- The C180 MLI is not directly available to user level NOS/VE programs. The C180 MLI interfaces will be used only by NOS/VE task services code.

- The C170 MLI is only available to C170 user jobs for message communication. Use of the C170 MLI for file staging and interactive communication will be done by C170 system privileged applications.

- Access to common MLI code from the C170 "trap handler" will not degrade C170 system performance significantly provided that all MLI code and data segments are "wired down".

- The C170 MLI subroutines which reside in the C170 program FL are considered to be part of the NOS/VE system in the same sense that the AIP subroutines are part of the NAM subsystem. NOS/VE will design, develop and maintain the C170 MLI.

- Synchronization between the C170 task and C180 tasks which use the MLI can be performed within task services code for each task. No MLI code will run in Monitor Mode and no additional monitor functions are required by MLI.

- A "wired down" shared segment can be used by MLI for tables and buffers. The MLI buffers will be an intermediate repository for messages transferred between C170 and C180. It is assumed that the overhead required to manage these buffers and to transfer data to and from them will not significantly degrade link or system performance.

3.6.5 DESIGN APPROACH

3.6.5.1 MLI Program Interfaces

The Memory Link Interface (MLI) is a set of program interfaces (both C170 and C180 versions will be provided) which allow a user or system application program to communicate with one or more other user or system application programs. MLI on C170 will use the CALL180 (017) instruction to call C180 code which manages communication through C180 memory (shared segment(s)).

MLI provides the following services for system or user application programs:

- Sign on or off from MLI (SIGNON/SIGNOFF).
- Add or delete permission for another application to send messages to it (ADDSPL/DELSPL).
- Determine whether or not a message can be sent to another application (CONFIRM).
- Send a message to another application (SEND).
- Determine whether or not other applications have sent messages to it (FETCHRL).
- Receive a message from another application (RECEIVE).

The following are general descriptions of the C170 and C180 MLI:

SIGNON (aname, maxmsg, status) [Application sign on request]

aname(input): application name.
maxmsg(input): maximum number of messages that can be received by this system application at one time.
status(output): request status.

SIGNOFF (aname, status) [Application sign off request]

aname(input): application name.
status(output): request status.

ADDSPL (aname, sname, status) [Add sender to permit list]

aname(input): name of application that "sname" will be permitted to send to.
sname(input): name of application that may send to "aname".
status(output): request status.

DELSPL (aname, sname, status) [Delete sender from permit list]

aname(input): name of application that does not want any more messages from "sname".
sname(input): name of application that will not subsequently be able to send to "aname".
status(output): request status.

CONFIRM (aname, dname, status) [Confirm that message can be sent]

aname(input): name of application wishing to see if it can send to "dname".
dname(input): name of application that "aname" wishes to find out if it can send a message to.

SEND (aname, dname, arbinfo, fwa, length, signal, status) [Send a message to an application]

aname(input): name of the application that wishes to send.
dname(input): name of the application that "aname" wishes to send a message to.
arbinfo(input): arbitrary information which will be associated with the message, but not part of the message, that can be quickly accessed by the receiver without reading the actual message.
fwa(input): first address of the message to be sent.
length(input): length of the message to be sent.

signal(input): indicates whether the receiver should be signaled to indicate that a new message is available for it.
status(output): request status.

FETCHRL (aname, sname, rladdr, status) [Fetch list of messages waiting to be received]

aname(input): name of the application for which messages may be waiting.
sname(input): sender application name - if given, information will be returned only for any messages from this particular sender - if not given, information will be returned for all senders.
rladdr(input): address of a buffer into which MLI will place information about message waiting to be received by "aname".
status(output): request status.

RECEIVE (aname, rindex, fwa, buflen, signal, msglen, arbinfo, status) [Receives a specific message]

aname(input): name of the application receiving the message.
rindex(input): index of the particular message to be received. See FETCHRL.
fwa(input): first address of the buffer that is to receive the message.
buflen(input): length of the buffer that is to receive the message.
signal(input): indicates whether or not the sender should be signaled after the message is received.
msglen(output): the length of the actual message received.
arbinfo(output): the arbitrary information associated with the message.
status(output): request status.

3.6.5.2 General Characteristics of MLI Interfaces

Some general characteristics of MLI communication are:

1. Sender and receiver do not both have to be in memory (swapped in) at the same time (i.e., both sender and receiver are permitted to swap out at any time).

2. The receiving application must "signon" before a sending application is allowed to "send" a message to it.

3. MLI is intended for C170-C180 communication but can also be

used for communication between tasks of separate C180 jobs or between separate C170 jobs.

4. All requests are synchronous (i.e., control is not returned until the request is complete). If the request cannot be completed immediately, an error status will be returned and the application must reissue the request. "Complete" does not necessarily mean that a message has been delivered to a receiving program - only that it is available to be received.

5. An application can send only one message at a time to a particular destination application (i.e., only one message can be in the "pipe" from the sending application to the corresponding receiving application). Note - this restriction does not prevent an application from sending to many different receiving applications or vice versa at the same time.

6. Use of MLI by NOS/170 programs requires that they be validated to use the CALL180 (017) instruction. This could be provided through a new user validation permission bit. Since NOS/BE does not provide user validation, the use of the CALL180 instruction can not be restricted.

7. System applications are allowed additional privileges that are not granted to user applications. System applications must be "loaded from system" (NOS/BE) or either SSJ=, subsystem, system origin, system origin privileges (NOS) - the latter is preferred for ease in testing, but which method is used will be determined for the most part by C170 Design.

   a) System applications can identify themselves with predefined application names - user applications are identified by system-assigned job/task name.

   b) System applications can specify the number of messages that can be sent to them at one time - the number of messages that can be sent to a user application at one time will be limited by MLI.

   c) System applications can specify that they are able to receive messages from "anyone" - user applications can only permit specific applkcations to send to them.

   d) The number of applications allowed to send to a user application (number of ADDSPL requests) will be limited by MLI.

e) System applications are allosed to "signon" and use several different application names at the same time.

System applications will be distinguished from user applications as follows:

On NOS/170, a system application must have system origin privileges. (Further discussion needed, as noted above.)

On NOS/BE, a system application must have System Library residence (i.e., losded from the system library flag is set in the control point area).

8. A SEND option to signal ("wake up") the receiver when the reciver is a C180 task is provided. Likewise, a RECEIVE ootion to signal the sender when the sender is a C180 task is provided. The corresponding options for C180 SEND to C170 and C180 RECEIVE from C170 will not be provided initially, but may be added later.

9. An "arbitrary information" field is provided on a SEND which allows the sender to pass control (or any other) information to the receiver. This field will be returned by FETCHRL so it can be examinwd prior to issuing a RECEIVE of the message. Some of the uses for the "arbitrary information" field might be:

a) NAM-type connection number for a terminal associated with a C180 interactive job.

b) Stream number for submultiplexed data transfers between two applications.

c) EOI or abnormal status indicators for a file transfer.

10. Redundant SIGNON and SIGNOFF requests are legal.

11. All messages must be less than or equal to a "maximum message length" which will be determined and enforced by MLI.

12. Code to perform MLI functions is package as follows:

- Code for all basic functions (accessing the shared segment) resides in NOS/VE Task Services of the task making the request.

- Code for C170 functions consists of:

a) MLI subroutines which reside in the FL of the application program (in the same sense that NAM AIP resides in the application FL).

b) C170 "trap handler" code which translates the MLI subroutine's CALL180 requests into NOS/VE Task Services calls and translates the response into a CALL180 response.

13. In order to avoid page faults in the C170 "trap handler", the NOS/VE Task Services code and the entire shares segment must be "wired down" to real memory.

3.6.6 NOS/170 DIFFERENCES

The C170 and the C180 MLI interfaces are internal system program interfaces which are not directly accessible to normal C170 or C180 user programs so external compatibility with NOS/170 is not required.

Although the MLI interfaces have some similarity with the NOS/170 UCP/SCP interface and with the NAM AIP interface, they are more general and symmetrical than either of them and therefore are not exernally compatible with them.

3.7 PERMANENT FILE MANAGEMENT

3.7.1 GENERAL RESPONSIBILITY

The permanent file functions provide mechanisms for managing files that are retained beyond job executions and system deadstarts. Permanent files may reside on RMS, magnetic tape or MSS media (R1 supports permanent files on RMS only). The functions include:

- Managing the registration of files in a catalog.
- Establishing job access to a permanent file.
- Access control mechanisms for defining and controlling file access by permitted users according to authorized access modes (e.g., read, write, or via specific program).
- Managing the migration of files to/from different storage media based on file usage, media usage and/or user selection. Users need not explicitly direct or manage the migration of permanent files to/from different storage media.
- Backup and recovery of permanent files.

The permanent file access control mechanism is a major element of the NOS/VE security and protection capabilities.

3.7.2 GLOSSARY

file owner: A file owner is defined to be the user whose identification is associated with the job in which a new file is established. All files have a single owner. The owner specifies and maintains the file's access control mechanism and user permissions.

catalog: A catalog is a system file that contains entries used to associate logical names of elements (e.g., permanent files, devices) with an element descriptor and access control list. The descriptor normally includes information describing the identification and location of the element. All permanent files are registered in a catalog. Each user is associated with a master catalog and may also create subcatalogs.

permanent file name: Each permanent file is identified by a permanent file name. It can be up to 31 characters in length and is a unique identifier relative to the catalog in which the file is registered.

file cycle: Multiple versions of a permanent file may be registered under one permanent file name. Each version is called a cycle and is uniquely identified by the combination of permanent file name and cycle number. All cycles of a permanent file share the same access control list. No restrictions are placed on the content or size of any cycle as each is a unique file.

access control list: Access control lists are the primary mechanism used to manage access to files. It is a list noting the identification of users who can access a file and how they may access it. The owner of a file maintains the access control list.

family: A family is a logical grouping of users, their capabilities and their permanent files. It is a logical unit that can be moved among physical mainframes without impacting users or their programs. The NOS/VE system initialization process or an operator command associates a family with a mainframe. A single mainframe can include one or more families. Multiple mainframe and link mainframe configurations include multiple families. The logical family name is used to locate and route information (jobs, files, messages, etc.) within a complex of mainframes.

3.7.3 DESIGN OBJECTIVES

- The permanent file system provides major features in support of the NOS/VE security objectives. This includes:

  Identification - Each permanent file and its owner are uniquely identified. An access control list is associated with a permanent file to identify persons permitted to use the file and their access privileges (need-to-know). A security level is defined for each permanent file. The file system depends on the login validation process for verifying a user identity.

  Controlled Access - Each request for use of a permanent file is governed by the access control list and an associated set

of access rules.

Surveillance - All successful or unsuccessful attempts to access a permanent file are logged.

- Reliability and recovery is a priority design parameter Catalogs can be automatically recovered as part of the system recovery process. Utilities support the backup and recovery of individual files or grous of files. Utilities are easy to use.

- Capabilities that allow use of working copies of a permanent file are provided.

- The cataloging mechanism is a general capability to be used for the registration and controlled access of permanent RMS files, permanent tape files as well as other system elements. This mechanism:

  - allows for user/installation selection of criteria for grouping things together in a catalog.

  - minimizes search times

  - minimizes interlock implications in large shared environments

  - supports a large number of entries

  - is easy to use

  - optimizes for the case of file creation and access by the file owner relative to the owner's default catalog

- The permanent file system supports the sharing of files within a single mainframe, within multiple mainframes, and within linked mainframe environments.

- The permanent file system supports files residing on RMS, magnetic tape and MSS media.

- Archiving of permanent files to/from RMS to/from magnetic tape or MSS media is provided.

3.7.4 ASSUMPTIONS AND CONSTRAINTS

- The access control mechanism is isolated into a single code module that executes at a privileged level. No "user program" can directly interface to the access control module.

3.7.5 DESIGN APPROACH

The permanent file system provides interfaces to:

- Register a new permanent file (DEFINE, SAVE)
- Establish direct access to a permanent file (ATTACH).
- Get a working copy of a file (GET).
- Replace a permanent file with a working copy (REPLACE).
- Remove a permanent file (PURGE).
- Change the file identification or other description attributes (CHANGE).
- Manage the access control permissions (PERMIT).
- Display catalog information (DISPLAY_CATALOG).

Permanent files are always associated with a family. Each request can include parameters that identify the family, file owner and the catalog under which it is registered. Normally, users are not concerned with these parameters as they access their own files within the environment established for them during the login process. The family parameter identifies a logical system and is used to determine the physical mainframe in which the family resides. This allows files or file data to be moved within a complex of mainframes without reprogramming or changing command statements.

Within a family, each authorized user has one master catalog that resides within the online storage associated with the family. The identification of a user's master catalog is recorded with the user's validation information. During the login process, the master catalog is made available.

User may also create subcatalogs used to conveniently manage various groupings of files. Subcatalogs are simply a special

file type and their identification and access control list is always registered relative to a master catalog.

Users authorized to create permanent files on auxiliary sets (removable storage not defined within a family's storage) register files or subcatalogs in a master catalog residing on the auxiliary set.

A catalog or subcatalog is a Byte Addressable file organization. Each permanent file entry consists of a basic record containing identification and general usage statistics and if needed, a variable number of additional records containing cycle descriptions, access control information and/or user usage statistics. The basic record contains all information neeeded to establish access for the owner of the file.

An access control list entry reflects an identification based on combinations of family, account, project and/or user names as noted below.

IDENTIFICATION				MEANING
F	A	P	U	User U in family F, account A, project P may access the file
F	A	P	-	Anyone in family F, account A, project P may access the file
F	A	-	U	User U in family F, account A regardless of project may access the file
F	A	-	-	Anyone in family F, account A regardless of project and user identification may access the file
F	-	-	U	User U in family F regardless of account and project identification may access the file
F	-	-	-	Anyone in family F may access the file

Each entry also states usage modes permitted for the associated identification and determines the "need-to-know". These include one or more of the following:

READ - The identification may read the file.

WRITE - The identification may write information starting at the beginning of the file or establish MODIFY or APPEND usage as defined below.

APPEND - The identification may add information to the end of the file.

MODIFY - The identification may replace, delete or insert information in the file.

EXECUTE - The identification may execute the file.

NONE - The identification is prohibited access to the file.

Over and above the access control list the file owner specifies a security level and ring brackets that also control file access. If the security level of the user requesting access is less than the security level of the file, the access is not granted even though the requestor has been granted permission in the access control list. (The user's security level is established at login and may vary from one session to another. The security level of the system may also vary and it controls which user may log in and at which security level.) Access is not granted if the ring brackets of the file do not match those of the requestor.

The owner of a file may also specify a password to be provided whenever a file is accessed.

Permanent file utility functions provide capabilities to:

- Obtain a backup copy of permanent files.
- Load a permanent file from backup storage.
- Generate statistical reports relative to permanent file catalogs, file usage and storage usage.

File utilities are used by NOS/VE to provide automatic archiving of permanent files.

3.7.6 NOS/170 DIFFERENCES

- A separate indirect file mechanism is not included in NOS/VE, therefore the user need not distinguish between direct and indirect files. The GET/SAVE/REPLACE functions are provided

and allow use of a working copy of any permanent file.

- Access control lists are associated with all permanent files. The semi-private cataegory that implies usage logging by specific user is an option for any permanent file. (In NOS/170, a semi-private category could not have access control.)

- The mode options are modified to satisfy C180 security objectives. WRITE mode in NOS/VE does not imply READ capabilities. Sharing options are specified as a separate parameter value.

- The permanent file commands do not include physical device parameters. These are specified with the REQUEST command.

- The packname terminology is changed to setname.

- Cycles and subcatalogs have been added.

- Access control is expanded to include account and project identifications. A security level attribute is added.

- PURGALL is not supported. At the command level, the user must request deletion of each specific file (PURGE) by name.

3.8 DEVICE MANAGEMENT

3.8.1 GENERAL RESPONSIBILITY

The Device Management function is responsible for the following:

- Controlling access to physical devices.

- Associating a file with a device.

- Scheduling of non-preemptible resources among jobs.

- Mounting/dismounting of removable media.

- Labelling RMS and magnetic tape devices.

- Set management.

- Managing the physical configuration.

3.8.2 GLOSSARY

Device Assignment - effecting the association of a NOS/VE file with a specific device.

Device Scheduling - an algorithm which allocates non-preemptible resources to jobs in a manner which prevents system deadlock.

System Deadlock - A phenomenon which occurs when two jobs have been assigned one or more non-preemptible units of the same resource, both jobs require additional units of that resource in order to complete their work, and there are no more units.

Set - A set is a logical unit of mass storage space. It can include one or more physical volumes of storage. Each set can contains one or more files. Any file can span volumes within the set but may not span sets.

Volume - A volume is a physical unit of external storage (e.g., disk pack, reel of magnetic tape, fixed head disk drive). Each volume is identified by a volume serial number recorded on the storage media.

3.8.3 DESIGN OBJECTIVES

- Device management will provide a command and program interface for advising the system of impending device requirements of a job.

- Device management will schedule devices among jobs to avoid system deadlock at device assignment.

- Device management will provide a command and program interface to enable assignment of a file to any file-oriented device supported by NOS/VE. This includes specific devices such as 7/9 track tape units, 844/885-1X/885-42 disk storage units, and terminals.

- RMS volumes may be directly shared between mainframes executing NOS/VE. RMS and tape controllers may be directly shared in the virtual environment by NOS/VE and a CYBER 170

system.

- RMS management functions support installation and user controls for assignment of storage space.

- Configuration tables are established at the time of system deadstart and may be modified during system execution.

- Dynamic reconfiguration of removable media is supported. Alternate paths to peripheral devices are supported and are used automatically by NOS/VE when required.

- RMS volumes can be grouped into a logical unit of storage to aid recoverability, usage control and transportability. The logical unit can be dynamically redefined using system utilities.

3.8.4 ASSUMPTIONS AND CONSTRAINTS

- NOS/VE R1 does not support removable RMS volumes.

- A basic Device Scheduler is provided in R1. A user may supply a single RESOURCE command located at the beginning of the SCL command stream (exact location to be fixed and to be determined). The job will not be selected for execution until all the devices required by the user are simultaneously available. A more dynamic device scheduling to be provided in NOS/VE R2 will remove these restrictions.

3.8.5 DESIGN APPROACH

3.8.5.1 User Interfaces

Two command/program interfaces control device scheduling and assignment. The REQUEST command/procedure associates a particular device with a file. Association of a device with a file is broken as the result of an UNLOAD or RETURN or when a file is closed.

3.8.5.2 Device Scheduling

The RESOURCE command/request directs scheduling of tape transports and disk drives among jobs. Each non-preemptible

resource (7/9 track tape, 844/885-1X/885-42 mass storage) has its own scheduling count which specifies the maximum number of resource units to be used concurrently by the job. The initial RESOURCE command/request establishes the scheduling count for each resource to be used. A user may change the scheduling count by issuing a subsequent RESOURCE command/request or a RETURN command/request. The scheduling count for a resource, once established, may be incresed by a job only if all files previously assigned to units of that resource have been returned to the system.

3.8.5.3 RMS Management

Mass storage volumes are grouped into logical sets. Each set is identified by a set name and each member volume is identified by a volume name (external and internal identifiers). Each set includes a master volume that must always be online when the set is in use. A descriptor of the set and a catalog directory is recorded on the master. The catalog directory identifies master catalogs residing within the set.

Based on usage and/or hardware characteristics, some sets are permanently online, some may be removable (note: R1 only supports online storage).

Each family is associated with one or more sets. NOS/VE automatically manages (e.g., space assignment, mounting, dismounting) family sets. Auxiliary sets (one that is not defined to the family) can also be accessed by validated users. Their use is directed by REQUEST commands/requests within a user's job.

3.8.5.4 Configuration Management

3.8.5.4.1 HARDWARE ELEMENT STATE

A hardware element dynamically assumes one of three primary states:

- On State

  The ON state indicates that the hardware element is assumed to be fully operational including:

  powered on
  media mounted
  controlware loaded

"ready"
etc.

Elements in the ON state may in fact not meet all of the above criteria and if not, appropriate action is taken by configuration management. This includes:

operator notification
controlware loads
CEM power sequences
etc.

An element whose state changes to ON is subjected to element dependent reinstatement procedures including:

controlware loads
CEM power on sequences
label searching

- Off State

The OFF state indicates that the hardware element is not available to the system.

An element whose state changes to off is "immediately" unavailable to any software access (including maintenance).

- Maintenance State

No assumptions are made about the condition of a hardware element in maintenance state. Only maintenance software can access these elements.

An element whose state changes to maintenance is immediately available only to maintenance software.

The following table illustrates the permissable hardware state changes from the point of view of the system operator, CE operator, and maintenance software. Maintenance software will only be able to be activated by a user who meets specified security measures.

3.0 FUNCTIONS
3.8.5.4.1 HARDWARE ELEMENT STATE

STATE CHANGE		PERFORMED BY		
FROM	TO	SYSTEM OPERATOR	CE OPERATOR	MAINTENANCE SOFTWARE
ON	OFF	Y	N	N
ON	MAINTENANCE	Y	N	Y
OFF	ON	Y	N	N
OFF	MAINTENANCE	Y	N	N
MAINTENANCE	ON	N	Y	Y
MAINTENANCE	OFF	N	Y	N

3.8.5.4.2 SYSTEM ELEMENT IDENTIFICATION

Every system hardware element has a unique identification number called the System Element Number (SEN).

Command interfaces to configuration management will normally utilize the SEN although physical path oriented references may be permitted in some commands.

References to hardware elements such as channels and controllers will also indirectly reference subordinate hardware elements.

3.8.5.4.3 CONFIGURATION DISPLAYS

Configuration displays provide information on any or all hardware elements. Displays may be based on SEN or SEN range; element class - e.g., disk, tape, controllers; hardware path - e.g., channel 10, equipment 7; and system state - e.g., ON/OFF/MAINTENANCE.

Display format and content include the following information (where applicable):

- physical connection data (paths)
- software defined status
- error history
- current physical state
- maintenance register content
- SEN
- device class/code (CEVAL based)
- current maintenance activity
- media info (mounted or not, set name, VSN, ring or noring, etc.)
- job related info (user name, last access, etc.)

3.8.5.4.4 CONFIGURATION MANAGEMENT COMMANDS

Operator commands are provided to perform configuration management functions. Some commands may be locked out of the command repertoire of specific operators such as CE's or tape room operators.

Configuration management functions provided by command are:

- Assign equipment
- Change a hardware element's state (ON, OFF, MAINTENANCE)
- Change a hardware element's software defined condition:
  - read only
  - no allocate
  - primary path selection
- Change description of hardware relationships (add/delete/modify configuration table info)
- Set programmable hardware registers
  - cache enables
  - map enables

3.8.5.4.5 VIRTUAL ENVIRONMENT PARTITIONING

The CPU is partitioned via the VMID field of an exchange package. The VMID, which is established via exchange or CALL/RETURN/TRAP, determines how the CPU is to fetch and interpret instructions and operands from central memory and how to interpret the register file and interrupts, etc.

Central memory partitioning between 170 and 180 is enforced via two hardware mechanisms. For CPU access, the virtual memory mechansim is utilized to map all 170 CM accesses into real addresses 0-N and all 180 CM accesses into real address (N+1) - (Memory size -1). For PP access, the IOU bounds register is used to disable PP CM write into accesses or one side or other of a real address N.

The memory link mechanism and the NOS/VE monitor is required to have limited access to CM in both states.

PP partitioning is enforced by utilization of the IOU bounds register. A PP is either part of the 170 system or NOS/VE and as such is only capable of writing into the appropriate state memory. PP's can be assigned from one system to the other at NOS/VE start up and drop.

Channels are software partitioned as required to access the 170 and 180 peripheral devices. The only channel which may be shared is the MCH.

Controllers may be shared but only if there is a dual channel access to permit dedicated channel access from each system.

Peripheral media on shared controllers are not shared between systems but are partitioned via software. This means that 844 controlware must access commands for both 170 and 180 formatted I/O. (64 x 60 bit secotrs and 256 x 64 bit sectors.)

3.8.5.4.6 MAINTENANCE INTERFACES

All configuration management requests made by maintenance software are processed by the CEVAL interface. Services provided by CEVAL include:

system maintenance interlock check
place an element in maintenance state
"assign" an element to a test/diagnostic job
"return" an element
reinstate an element (from maintenance state)

The hardware elements referenced by the interface can be described via SEN or via physical path. A physical path reference need not correspond to an actual entry in the system configuration tables.

Additionally, where there is a reference to a controller or channel element, all subordinate elements are implicitly referenced.

CEVAL device codes will be tied to system defined device codes.

3.8.5.4.7 CONFIGURATION DEFINITION

The method for defining a system configuration is the Configuration Definition Language (CDL). CDL statements will be processed by a CDL processor available during NOS/VE deadstart (at basic system run time) and later during normal system run time. CDL statements may be entered via operator keyin or presented to the CDL processor in a file. CDL provides a means for establishing all configuration table information.

A complete multimainframe configuration can be defined with a set of CDL statements which can be presented to each NOS/VE system in the MMF configuration. There is a hierarchy of CDL statements which reflects the hardware paths of a configuration:

```
mainframe statement
   IOU statement
      channel statement
         controller statement
            unit statement
            unit statement
                  .
                  .
                  .
         controller statement
      channel statement
      PP statement
   IOU statement
         .
         .
         .
mainframe statement
         .
         .
         .
```

3.8.6 NOS/170 DIFFERENCES

- NOS/VE REQUEST capabilities differences include:

  - NOS/VE allows a user to associate a file with a particular mass storage set and/or set members.

  - NOS/VE does not support the ASSIGN command of the NOS/170. A subset of the devices supported on the ASSIGN command are supported by NOS/VE REQUEST. Specifically, these are:

    885-1X/885-42
    844-4X (DJ)
    terminals (TT)

    Device mnemonics for mass storage equipment are changed to be more descriptive of the hardware devices.

  - NOS/VE will support devices which will not be supported by NOS/170 (such as four-head parallel FMD).

  - NOS/VE system supports assignment of contiguous areas of mass storage to files. The user can specify the size of the contiguous area (allocation unit size) via REQUEST.

3.0 FUNCTIONS
3.8.6 NOS/170 DIFFERENCES

- NOS/VE RESOURC capabilities differ from NOS/170 with respect to specific non-preemptible resource support. NOS/VE will not support many of the devices supported by C170. Plus, NOS/VE will support new devices which will never be supported on C170. However, the NOS/VE RESOURC scheduling algorithm will be externally compatible with NOS/170.

- NOS/VE will assign a tape unit to a file when the file is opened rather than at the time the REQUEST command is encountered.

3.9 SEGMENT MANAGEMENT

3.9.1 GENERAL RESPONSIBILITY

- Providing segment management services for a task.
- Providing segment management services for job management, program management, I/O and maintenance services.
- Assigning the Active Segment Identifier (ASID) to a segment.

3.9.2 GLOSSARY

Address Space - The set of segments addressable in a task. Each address is uniquely identified by a segment number and a byte number.

Known Segment Table (KST) - A table indexed by segment number which contains a pointer to the segment's FAT and segment interlock record (SIR).

Segment Interlock Request (SIR) - A record located in virtual memory which is used by segment management to coordinate segment sharing and to store segment attribute information.

3.9.3 DESIGN OBJECTIVES

- The segment is a major element of system security. For this reason, a secure subset of segment attributes are externalized to user programs. Hardware structures such as the ASID, segment descriptor and the segment descriptor table are not externalized.
- Segment management provides a collection of internal system interfaces which are not accessible from user rings. These system interfaces satisfy the requirements of job management, program management, buffer management, loader and maintenance services.

3.9.4 ASSUMPTIONS AND CONSTRAINTS

- Cumulative lengths of segments allocated to a task are constrained by the amount of mass storage authorized to the job.
- Shared segments with write access must have the cache-bypass attribute due to multi-processor considerations (each processor has its own cache).
- Sharing of file segments between jobs can be accomplished via permanent file functions (ATTACH).
- Code segments cannot be passed between tasks. However, code in a NOS/VE library can be accessed as a shared file segment.
- Data segments may be passed between tasks of the same job. Such segments are limited to read and/or write access.
- Creation of binding section segments is the province of the NOS/VE loader only.
- Binding section segments cannot be passed or shared.
- Binding, execute and write access attributes are mutually exclusive. Tasks sharing a segment may not violate this rule.

3.9.5 DESIGN APPROACH

Since the segment is a basic element of NOS/VE system security, it is necessary to prevent unauthorized access to the segment descriptor content. To this end, two interfaces will exist, program and system.

The program interfaces to segment management deal solely with temporary segments. A second interface, unavailable to end-user, allows system control of segment descriptor entries.

3.9.6 NOS/170 DIFFERENCES

- The conflict over user and system extension of user's field length which was introduced in NOS/170 by the implementation of the Common Memory Manager (CMM) has been eliminated in

3.0 FUNCTIONS
3.9.6 NOS/170 DIFFERENCES

NOS/VE. CMM will be replaced in NOS/VE by a combination of PASCAL-X memory management features and the use of temporary segments by both user and product set. User and product set can create unique, extensible segments without conflict.

3.10 SYSTEM ACCESS

3.10.1 GENERAL RESPONSIBILITY

The system access function is responsible for controlling access to the system and, further, to selected hardware and software facilities of the system. Access is granted or denied to users (people) and the programs that execute on their behalf. The controls may function to allow free access, limited access or no access. These controls are established and maintained by a hierarchy of users functioning as "administrators". The purpose of the access controls is to benefit installation management, user management, and users.

3.10.2 GLOSSARY

- Account: in the simplest terms, a "charge account" - users may be permitted to charge system resource expenditures to an account.
- Account Administrator: an administrator who may create, delete and alter descriptions of projects within the relevant account.
- Administrator: a user who has been given certain privileges of control over the system access capabilities of other users.
- Member: a user permitted to charge system resource use to a given project.
- Project: a subdivision of an account that may be used by the account administrator for bookkeeping and resource allotment purposes.
- Project Administrator: an administrator who may create, delete and alter descriptions of members within the relevant project.
- System Administrator: an administrator who may create, delete and alter user descriptions and account descriptions.
- System State: a name for one of any number of arbitrary

system states - the installation may use this name to denote the type of system access allowed - current system state is selected by the system operator (or by system programs).

- User: an individual known to the system by "user name".

3.10.3 DESIGN OBJECTIVES

The system access function will provide or assist in the provision of protection, identification, and access visibility.

- Protection

  - of users from each other - assist in the equitable allocation of resources among system users
  - of the installation resources from over-use - for examole too many tape mounts; requests for mounting more disk packs than there are available drives
  - of the installation from unauthorized accrual of charges
  - of accounts from cost overruns
  - of users from themselves and the jobs they run - to avoid cost, time, and resource usage overruns

- Identification - to assure all users of the system are uniquely identified and use of the system is identified with a particular user for purposes of:

  - protection
  - security
  - file and job privacy and security
  - billing

- Visibility

  - allow users and authorized administrators to view the limits and characteristics associated with them or with the system, account, project, or project member(s) for which they are responsible
  - allow users and authorized administrators to view their current cumulative state or progress relative to appropriate limits

3.10.4 ASSUMPTIONS AND CONSTRAINTS

- Each system user must be uniquely identified. It will be assumed that multiple people do not share a common user identification.

- Provision for "visitor"?

- All system use must be accountable or billable. Therefore, whenever the system is being accessed by a user, there must be an authorized account to charge.

- Limits are placed primarily upon what an individual user does for a project, not upon the user. Willingness to pay and need for resources are associated with the work a user does, not the user.

- System access characteristics (authorized users, limits, etc.) must be manipulable - in a controlled fashion - by people other than centrally located installation personnel:
  - creation, deletion, of user descriptions
  - setting limits on users, accounts, ...
  - setting defaults for users, accounts, ...

- All user, account, project, etc. description records must provide elements reserved for installation use.

- Provision must be made for an installation-defined module to be executed at:
  - job entry time - before the job is queued (if appropriate)
  - beginning of job execution - after coming out of an input queue
  - any time certain resources are requested (restricted access programs, magnetic tapes, disk packs)

3.10.5 DESIGN APPROACH

3.10.5.1 User Management

The fundamental groundrule under which these user definition and management facilities were designed was that they were to be "NOS-like". Therefore, the current design used by NOS was assumed to be the base and additions, deletions and alterations were made for the purposes of resolving existing NOS RSMs, other suggestions, and changes in operating environment due to CYBER 80

such as the basic hardware and software architecture.

The NOS/VE user management scheme assumes, as does NOS to a lesser extent, that each individual person who uses the computer system has a unique user identification. In NOS this is called the "user number". In NOS/VE it is called the "user name". (The change in terminology reflects a desire to humanize the interface a bit). The design of NOS/VE attempts to facilitate this unique identification by such things as making it easier to create new users, and by enhancing the convenience of sharing permanent files among groups of users.

NOS has two levels of people who exercise some level of control over user descriptions. The highest is an analyst who may create new users, delete users and alter user validation information. For each charge number, a "master user" may be appointed to add and delete users from projects and to maintain certain limits on use of the system and of account funds by individual project members. NOS/VE extends this two level hierarchy by one level by allowing "account administrators" (rather like master users) to appoint "project administrators" who can exercise some control over project members. Although only three levels are defined here, if the need arises the basic design can be extended to cover more. Also, through the use of preset default values, users may be made generally unaware of any hierarchy. This would be useful to installations that do not wish to break their users into projects and to charge by account and project.

Two fundamental assumptions are: 1) all system use must be accountable;) limits should be placed upon what an individual does for a project, not upon the individual user. The first assumption means that, for example, all jobs, all connect time and all permanent files must be chargeable to a particular account and project. This means that the user must be "running under" a project at all times. The second assumption means that the account and project administrators have more control over users than is allowed by NOS.

3.10.5.2 Administrators

The concept of "administrator" is used in order to provide a focal point for responsibility, control and accountability. An administrator is responsible for the use of the system by some group of users. The administrator can be held accountable for certain aspects of this use. These responsibilities require that the administrator have some level of control over the users in the group when they are doing group-related work. NOS/VE

provides several levels of administrators in order to facilitate this control. An administrator may exert controls directly - without the need to get some other person to do so.

Each administrator may place restrictions on all "subordinate" activities. For example, the System Administrator (SA) may specify that account "Plumco" may have no more than 20 permanent files. The Account Administrator (AA) of "Plumco" may then specify that the subordinate project "Shipping" may have 5, project "Manufacturing" may have 10 and project "Receiving" may have 5. The Project Administrator (PA) for "Shipping" may then place a limit of 2 on project member "J_Jones", 2 on "M_Smith" and 1 on "Linda_Davis". The "Shipping" project may never exceed 5 files, no matter how many members are included in the project by the PA. If this were desired, the PA would have to make arrangements for the "Plumco" AA to increase the limit on the project (which would then probably cause a reduction for some other "Plumco" project).

Each of the major entities - accounts, projects, members and users - are characterized in seperate "descriptions". Each description is "owned" by the appropriate administrator. The SA owns the accounts-description, the AA owns the projects-description, and the PA owns the members-description. In addition, since individual users may be members of multiple projects and multiple accounts, the SA owns the users-description. Access to these descriptions is controlled by a program or programs which have the ability to allow certain users to operate on the description data in certain ways. For example, the PA(s) would normally be permitted by their AA to examine (only) the limits on their projects and to examine and alter the default ring number for their projects.

3.10.5.2.1 SYSTEM ADMINISTRATOR

The SA is a registered user. There may be several SAs per installation, one (or more) for each family - or the same individual may be registered in each family and perform the SA function for accounts within that family. The SA in this context defines, undefines and alters the descriptions of all accounts, including AA identification, limits and other account characteristics. The concept of "account" is for the benefit of the installation management. This management needs to know who to charge for services rendered, resources consumed, etc. Limits are placed upon accounts in order to provide control over system use. This facilitates effective installation management.

3.10.5.2.2 ACCOUNT ADMINISTRATOR

The AA is a registered user. Each AA is appointed by the SA and need not be a member of a project within the account. There may be more than one AA per account. The AA defines, undefines, and alters the descriptions of projects within the AA's account. The concept of project is for the benefit of the AA. Privileges and charge reports are provided by project in order to facilitate account administration.

3.10.5.2.3 PROJECT ADMINISTRATOR

The PA is a registered user. Each PA is appointed by the AA and need not be a member of the project to be administered. There may be more than one PA per project. The PA defines, undefines, and alters limits and other characteristics of project members. A project member is a user who is allowed to charge system usage costs to a particular project. Limits are placed upon members rather than users since type of work and ability or willingness to pay depends upon a project - not a user. [There is no reason why a user could not be the sole member of an individual account and project.]

3.10.5.3 Account, Project, Member and User Descriptions

3.10.5.3.1 GENERAL

Element Access Control

Each account, project, member and user description includes enforced limits imposed by an administrator and default values for the convenience of the respective user or group. The imposed limits should be protected from alteration by the "limited" entity. The defaults, however, should be alterable. In both cases, it is necessary to be able to examine the current values.

To allow this, each element in each description has associated with it a change level. Each level has an associated value. These values are set by an administrator and limit whether a subordinate can change the element. In order to change element, the subordinate must be running at a level greater than or equal to the value. For example, the SA may set the "change" value to 99. If the defined running levels for users are always between one and 90, then the AA would never to allowed to alter it.

The exact change levels and method for subordinates to be assigned these running levels will be described elsewhere as part of the general support for element security as applied to

operating system data.

System State

The concept of "system state" is present to allow installations to control the type and amount of use the system gets. Some system states might be, in any rational combination:

- Maintenance mode.
- External customers only.
- "Top secret" operation.
- Prime time.
- No operators.

The system state is represented by a value that may be altered by an operator. The values (at least at this point in design) are entirely arbitrary and serve only to identify which set of operating conditions are to be enforced - such as those in the "system state 'matrix'" below. The characteristics and limits in this 'matrix' are associated with system state because they deal with what a single job may do - and an installation may wish these to be different from one system state to another. For example, during "top secret" system state, only "access paths" from secure locations by "top secret" cleared users would be permitted. Or, when there are no operators, zero magnetic tape units could be used at one time.

3.10.5.3.2 DESCRIPTION ELEMENTS

Account, Project and Member Descriptions

- Name

"Name" must be unique within the next higher entity. That is, account name must be unique within family, project within account and member within project. For member, this element contains a registered user name.

- Administrator Names

These are the registered user names of the administrators. This element is not meaningful for member description. Associated with each administrator name is information regarding the modes of permanent file access that the administrator will automatically have to all permanent files of all subordinate members. These modes may not be overridden by the members (for example, via PERMIT or CHANGE commands). This specification allows modes of access to be given as null, meaning no automatic access privileges.

- Subordinates - Maximum Number

The maximum number of projects within an account and members within a project is specified here. This element is not meaningful for member description.

- Subordinates - List of Names

For an account description, a list of project names appears here. For a project description, a list of member names is given. This element is not meaningful for member description.

- Defined But Not Usable - Switch

A complete description may exist for an entity without the ability to actually use the entity. If this switch is "on", the name is reserved and all information is kept, but the account, project or member cannot be validly used.

- Valid Date Range

Use of the entity is restricted to dates within this range. Outside of the range, the entity remains defined, but is unusable.

- Security Count - Maximum

This element specifies the maximum number of security violations that are allowed each entity member.

- Permanent Files - Maximum Number

The entity may have this many total permanent files cataloged on public devices. Requests to create files in excess of this limit are rejected.

- Permanent Files - Maximum Total Size

The entity may occupy this much mass storage space for all permanent file storage on public devices. Requests to allocate storage in excess of this limit are rejected.

- Permanent Files - Maximum Individual Size

No individual permanent file within the entity may be larger than this size. Requests to allocate storage in excess of this limit are rejected.

- Prolog Procedure - Forced

The SA may, for example, want all logins for a particular account to run a comprehensive security check before allowing processing. This procedure is specified here. Any set of valid command may be contained here. They are automatically executed before any user-specified commands. If account, project and member descriptions all specify this element, then the order of execution is:

1) Account prolog - forced.
2) Project prolog - forced.
3) Member prolog - forced.
4) User processing which may include an elective prolog.

Although the term "login" is used, the procedure will be activated for all job modes.

- Epilog - Forced

This set of commands is the inverse of the forced prolog. It is automatically activated at logout time - whether normal or abnormal. The order of execution is the reverse of the prolog sequence. No user or procedure action can cause "forced" epilogs to not be executed.

- Installation-Defined Limits, Characteristics

Installations may define additional elements.

- System State "Matrix"

For each system state, values are assigned for the following elements.

• Access Paths Allowed

This element specifies the physical paths that jobs may use to enter the system. For example, "system console", "lines 29-45 on network processor 3", "answerback drum 'xyz'", "local card reader 3", etc.

• Family Access Allowed

This element specifies the families that entity jobs may use for purposes of job execution, output file disposition, etc.

• Magnetic Tape Units At One Time - Maximum Number

Each entity job may have no more than this number of magnetic tape units in use at one time. Requests for more than this number are rejected.

- Removable Packs At One Time - Maximum Number

Each entity job may have no more than this number of removeable packs in use at one time. Requests that would exceed this limit are rejected.

- Messages Per Job - Maximum

Each entity job may issue no more than this number of MESSAGE requests to the system dayfile. Jobs which exceed this amount are terminated. Interactive users are warned prior to the limit being exceeded.

- Batch Control Statements Per Job - Maximum Number

Each entity job may execute no more than this number of control statements in batch mode. Jobs which exceed this amount are terminated.

- Print/Punch Files Disposed At One Time - Maximum Number

Each entity job may dispose no more than this number of files to print and punch queues. Requests which would exceed this amount are rejected.

- Cards Per Punch File - Maximum Number

Each entity job may punch no more than this number of cards per punch disposition file. Files which exceed this number are not punched in their entirety.

- Lines Per Print File - Maximum Number

Each entity job may print no more than this number of lines per print disposition file. Files which exceed this number are not printed in their entirety.

- Central Memory Per Job - Maximum Words

Each entity job may use no more than this amount of real memory at one time. Job requests which exceed this are rejected.

- Bulk Storage Per Job - Maximum Words

Each entity job may use no more than this amount of bulk memory at one time. Job requests which exceed this are rejected.

- SRUs Per Job - Maximum Units

Each entity job may consume no more than this number of SRUs. Jobs which exceed this are terminated. Interactive users are warned prior to the limit being exceeded.

- Files At One Time Per Job - Maximum Number

Each entity job may have no more than this number of files known locally to the job at one time. File requests which would exceed this number are rejected.

- New Mass Storage Allocated Per Job - Maximum

Each entity job may use no more than this amount of new storage per job. The limit is checked dynamically. Any file request which would exceed this limit is rejected.

- Deferred Batch Jobs in Queues At One Time - Maximum Number

No more than this number of entity deferred batch jobs may be present in queues at one time. Requests to activate new deferred batch jobs are rejected if this limit would be exceeded.

- May Request Non-Allocatable Equipment - Switch

This element specifies whether users of this entity may use non-allocatable equipment requests. If not, and such a request is issued, it is rejected.

- May Issue Auxiliary Device Requests - Switch

This element specifies whether users of this entity may use auxiliary devices. If not, and an auxiliary device request is issued, it is rejected.

- List in "Online Users List" - Switch

The "online users list" contains the names of users that are currently running in the system. If no entity member may ever appear in the list, the switch is set to "never". If all entity members must always appear, the switch is set to "always". If the entity member may choose to appear or not, the switch is set to "toggle".

- Ring Number - Minimum

No entity job may specify a ring number lower than this. Requests which would violate this limit are rejected.

- Ring Number - Default Initial

The default initial ring number for a job is specified here. This number must be greater than or equal to the "ring number - minimum".

- SRU Coefficients (Account Only)

Each account may be assigned different SRU coefficients by the SA with this element.

- Terminal Output Access Control List

This element defines what users or members may dispose output files to terminals being used by entity members. These permissions are similar to those applied to permanent files. For example, in a member description, this element may specify that when this member is logged in at a batch terminal, the terminal may only recieve output files from:

anyone in project A
and anyone in project B
and user "smith"
and user "jones" when "jones" is running as a member of project C.

- Job Classes Permitted

Each entity job may use only these job classes. Jobs which specify non-permitted classes are rejected or terminated. [Job classes may include such things as priority, maintenance job, system job,...].

- Job Class - Default Initial

This element specifies the default initial job class for all entity jobs.

- Job Types Permitted

Each entity job may use only these job types. Jobs which specify non-permitted types are rejected or terminated. [Job types may include such things as transaction,...].

- Job Type - Default Initial

This element specifies the default initial job type for all entity jobs.

• Job Modes Permitted

Each entity job may use only these job modes. Jobs which specify non-permitted modes are rejected or terminated. [Job modes may include such things as interactive, batch,...].

• Installation-Defined Limits, Characteristics

An installation may specify additional elements here.

User Description Elements

- Name

Each user has a unique name within the family. This name is used to identify the user to the system and to other users.

- Uniqueness Guarantor

Since several individuals may create new users, and since it is important to have only one record of an individual user, some unique value is needed to assure only one record. This element may contain a value that is inherently unique such as employee number. When a new user is created, this element can be scanned for all users to check for a duplicate.

- Password

This password verifies the user to the system.

- Account and Project Name - Initial Default

Since each user must be charging to a project at all times, a method is needed to specify the project to be charged at least from the time of login to the time of an account and project specification command. This default provides this. In addition, it simplifies the user's interface if he always or usually uses only one account/project.

- Address Information

Miscellaneous information about the user's mail address, etc.

- Prolog - Elective

This element specifies the set of commands to be used at

login. These commands are chosen by the user and are automatically executed after any forced account or project prologs.

- Epilog - Elective

Any logout commands that the user wishes to execute on a normal or abnormal logout is specified here. They are automatically executed before any project or account epilogs.

- Last Login Date and Time

As a security precaution, the date and time of the user's last use of the system are recorded here for the user's examination.

- Security Characteristics

Up to ten user characteristics may be placed here in order to provide verification of the user over and above the simple password. For example, drivers license number, mother's name, etc.

- Installation-Defined Characteristics

Installations may define additional characteristics.

3.10.6 NOS/170 DIFFERENCES

- "access permission" is removed - capabilities of "access":
  - "dial" command - replaced by generalized terminal-to-terminal communication capability
  - "monitor" command - replaced by generalized terminal-to-terminal communication capability - the "wiretap" ability of "monitor" is removed
  - "user" command - replaced by generalized "online user list"
- "default character set" removed - character set is ASCII
- all aspects of indirect/direct permanent file limits are merged
- the concept of "user index" is removed
- charge, project is always required
- "special transaction privileges" removed - replaced by "job

modes permitted"

- "permission to use System Control Point" removed - no System Control Point

- user can always change his own password - users do not share user names, user needs privacy, controls may be exerted on members by SA, AA, PA

- "system origin job privileges" replaced by job class concept and "job classes permitted"

- "may access system files (library)" replaced by permanent file capabilities including access by catalog

- "special accounting privileges" replaced by the hierarchy of administrators and the associated limitations and capabilities of administrators

- default terminal characteristics, "initial subsystem", "timeout" replaced by ability to insert corresponding commands into prologs

- three levels of administrators - SA, AA, PA - replace the "master user" and "system analyst" concepts and capabilities

- "may create permanent files" replaced by ability to set "maximum number of permanent files" to zero

- "answerback" replaced by "access paths allowed"

- "CPU time per job step" removed. This may be specified on a job by job basis by user parameter or command. SRUs per job is felt to be a sufficient limit.

- automatic read-only permission to permanent files that is given via the presence of asterisks in a NOS user number is replaced by the ability to specify automatic, forced accessibility modes associated with administrators.

- "terminal access control list" expands upon the NOS capability of only allowing output to a terminal logged in by the same user (or to subordinate via asterisks in user number). The ability to limit what will appear at a terminal is retained, but accomodation is made for public terminals, project-only terminals, etc.

3.11 SYSTEM LOGGING

3.11.1 GENERAL RESPONSIBILITY

The system logging function is responsible for maintaining several system logs for the purpose of recording system and job activities in order to provide:

- data for analysis of system performance
- data for analysis of system workload
- detailed job-step level trace of job flow
- data for analysis of hardware performance and usage
- trace of system operator activities
- billing information

Each system log has a specific set of uses and access to a specific log is controlled.

3.11.2 GLOSSARY

System Log - The system log is a repository for information regarding external system workload. That is, the work the system was asked to do via commands and the high level responses of the system in regard to the commands.

Account Log - The account log contains accounting and billing information. This consists of resources and/or services used, "who" used them and "who" to charge. The account log should be the only log needed for an installation to do billing.

Engineering Log - The engineering log contains information regarding system hardware usage and errors. The engineering log should be the only log needed to perform hardware usage and error analysis.

Statistics Log - The statistics log contains detailed system workload information and detailed system performance information (i.e., the way the system responds to the workload).

Although some of this information is recorded in other logs, a separate log is maintained in order to:

- keep otherlogs relatively "clean" or oriented to their own purposes
- allow possibly large amounts of data to be recorded in a compact binary form

It is intended that the installation have flexibility in deciding what is to be logged and when logging will occur.

Job Log - The job log contains a trace of job execution. Information concerning the work requested and accomplished is recorded here. It provides a summary of the flow of the job, problems encountered and charges accured by the job.

Job Statistics Log - The job statistic log content is similar to that of the global statistic log. It contains detailed job performance information. The recording of information in the global log is controlled by the installation in regard to what is recorded and when. For the job statistic log, this control is exerted by the user. Although the installation may determine what may be recorded in this log, the user determines when logging is to occur.

3.11.3 DESIGN OBJECTIVES

To be supplied.

3.11.4 ASSUMPTIONS AND CONSTRAINTS

To be supplied.

3.11.5 DESIGN APPROACH

Global logs are maintained for the entire system. All jobs may make entries into these logs. Only one of each is defined at a time - as opposed to job-local logs where there will be many occurrences, one per job.

Global logs are permanent files owned by the system, and as such, receive the same protection and recoverability capabilities

as all other permanent files. In order to identify the particular permanent files which are the logs, there will be a catalog for the "system". At system recovery time, the logs are ATTACHed via this catalog using predefined names.

The job-local logs are created as temporary files for each job. Protection and recovery methods available for local temporary files are available for these logs. A special case may be that of the operator facility (facilities?) which will execute as a job. In this case, it is important to not lose the record of operator activities after interruption. If this is not fully handled by job and temporary file recovery, then these logs will have to be created and recovered as permanent files.

In order to assure that the individual global logs are stored on devices with characteristics appropriate for each log, the installation will be able to specify the set name upon which each log is to be recorded. System initialization allows this to be set. When the system is initialized, these files wil be DEFINEd on the proper sets.

3.11.5.1 Account Log Content

The account log information includes:

- Date whenever it is established or it changes.
- All system initialization and recoveries
- Entry of a job into the system.
- Start of each job execution.
- End of each job execution.
- End of each job execution.
- End of disposition of each output file.
- Each rerun of a job.
- Establishment or change of project membership for a job.
- Establishment or change of SRU coefficients for a job.
- All accounting accumulators at end of job and whenever project membership for a job changes and whenever SRU coefficients change.
- All accounting information items that have no accumulators - recorded as services are rendered.
- Others as determined by accounting design.

3.11.5.2 Engineering Log Content

The engineering log information includes:

- Date whenever it is established or it changes.

- All system initializations and recoveries.
- Hardware configuration (e.g., using NOS/170 terminology, which CMRDECK was used).
- Hardware configuration options and changes exercised at deadstart and during operation
- Identification of controlware loaded.
- Identification of version, etc. of peripheral drivers, MCU software and the "maintenance job".
- System name, version, etc.
- Abnormal occurrences relating to hardware.
- Hardware usage information (cards read, lines printed, tape blocks written...)
- Mainframe identification and options.
- Others as determined by physical I/O, ESS, etc.

3.11.5.3 Job Log Content

The job log information includes:

- Date at beginning of job and whenever it changes.
- All commands and command error messages.
- Any major changes in the job's status in the system (e.g., start of reading card deck, end of reading card deck, placed into input queue, moved to mainframe x, dropped by operator).

Each entry contains these fields:

FIELD	SOURCE OF FIELD
DATA	
Time of day to millisecond	Task services
Origin of message	Task services
Message text	Caller

3.11.6 NOS/170 DIFFERENCES

To be supplied.

3.12 SYSTEM ACCOUNTING

3.12.1 GENERAL RESPONSIBILITY

The system accounting function is responsible for measuring system use by individual jobs in order for the installation management to assess charges for this use. System access controls are provided for both installation and account management in order to avoid cost overruns. The set of chargeable activities (e.g.CP time, memory) is selected by the installation and a function for combining these items into a single billing unit is defined. Methods to examine and limit rate of expenditure of the billing unit and its components are provided.

3.12.2 GLOSSARY

To be supplied.

3.12.3 DESIGN OBJECTIVES

NOS/VE accounting provides:

- Consistent accounting information for each execution of the same process in the same environment.
- A single billing unit that reflects all charges accrued by a job.
- Detailed information relative to system usage. The single billing unit is a function of this set of data. This information is available to users and installation personnel to support charges.
- The current total of the single billing unit and of the detail system usage totals is available to users at any time within a job.
- Installation options allow tailoring of whch system (includes application product) resource usage, events or services comprise the billing unit (e.g., CPU seconds, records read, memory used). In addition, NOS/VE allows tailoring of the

relative weights of each datum used in the billing unit algorithm.

- Accounting information for resources or services whose use is controllable by the user is available in "user terms". Examples include number of statements compiled, number of files accessed, service level used, number of linear equations solved.

- Accounting information that reflects allocation of vendor costs to users is available in "cost recovery terms". Examples include CPU seconds used, bytes of memory used, channel seconds used, disk sectors used. These units are not always understood or controllable by users.

- Support of a hierarchy of "accounts" (charge numbers in NOS/170), projects within accounts and members (users) allowed to charge to particular accounts. For each account an administrator controls which users may charge to that account and controls usage of authorized funds by individual users.

- Support for billing and inter-installation cost recovery in multi-computer networks.

- Support for "application accounting" which allows (under controlled conditions) applications to "unit price" their services (e.g., charge for number of plots produced rather than for the resources used to generate the plots) - and allows (under controlled conditions) applications to alter the algorithm used to compute the billing unit.

3.12.4 ASSUMPTIONS AND CONSTRAINTS

To be supplied.

3.12.5 DESIGN APPROACH

To be supplied.

3.12.6 NOS/170 DIFFERENCES

To be supplied.

3.13 OPERATOR COMMUNICATION

3.13.1 GENERAL RESPONSIBILITY

The operator communication function is responsible for permitting communication between system components and various system operators and between user jobs and operators. Displays of system activity and requests for operator action are generated. Commands from the operators are processed to control system operation.

3.13.2 GLOSSARY

CC545 - The C170 operator console for NOS/170 and NOS/BE. It will not be the standard operator console for NOS/VE but can be used as an alternate NOS/VE operator console through the C170 for NOS/VE R1 or as a directly driven console by NOS/VE native mode in a later release.

Command/Display Processor - System provided programs which execute as normal NOS/VE user level programs to provide the operator command and display interface.

Display Building Procedures - A set of procedures which are called by the Command/Display Processors to structure information that is to be displayed on the display screen.

Display Generator - A procedure which is called by the Command/Display Processor to format display information according to the characteristics of the Operator Console and send the formatted output to the display screen.

Operator Console - Any console or terminal which can be used by a NOS/VE operator to provide visibility and control of the NOS/VE system. It may be a 752, a CC545 or a NOS/VE interactive terminal.

Operator Facility - The collection of NOS/VE procedures which provide the capabilities required for communication with a System Operator.

Operator User - A NOS/VE user who is granted System Operator

privileges by the NOS/VE validation mechanism.

Standard System Console - A 752 terminal which is connected to NOS/VE through the Two Port Mux.

System Job Interface Procedures - A set of Task Services and Monitor functions which allow the Command/Display Processors to pass information to or from NOS/VE system jobs in order to provide visibility and control over system jobs.

System Interface Procedures - A set of Task Services and Monitor functions which allow the Command/Display Processors to obtain information or exercise control over jobs, tables, etc. in the NOS/VE system.

System Operator - A privileged interactive NOS/VE user who is allowed to display and control some portion of the NOS/VE system through operator command and display requests. An installation will normally assign operator privileges so that one operator is designated as the "master operator" - all other operators would be "auxiliary operators" with less privilege than the "master operator".

System Operator Jobs - The interactive NOS/VE jobs which are associated with privileged interactive NOS/VE users who are validated as System Operators.

Two Port Mux - A hardware multiplexer connected directly to the C180 which has one port for the Standard Operator Console and the other port for the Maintenance Console.

Unattended Mode - A mode of NOS/VE operation in which the system functions automatically without human operator intervention.

3.13.3 DESIGN OBJECTIVES

The general objective of the NOS/VE Operator Facility is to provide the link between NOS/VE and System Operators. The primary functions of System Operators are to respond to requests for human intervention by the NOS/VE system and to exercise control over how the system operates. In order to perform these functions properly, System Operators must be able to request that the operating system display information associated with these

operator functions.

More specific objectives of the NOS/VE Operator Facility are to provide the following types of capabilities:

- Provide status and control of the hardware components of the NOS/VE system such as memory, I/O channels, PPUs, disk units, tape units, unit record equipment, communication equipment, etc.

- Provide status and control of NOS/VE jobs which are in execution or in input/output queues and also the resources associated with these jobs (memory, disk space, files, tape drives, disk drives, terminals, etc.).

- Provide status and control for NOS/VE system components such as the basic operating system, system jobs, special installation applications, etc.

- Request specific actions from the System Operators such as mount a tape, put paper in the printer, and accept responses from the System Operators to these requests.

- Dynamically provide visible information to the System Operators so that they can easily determine that the system is functioning properly.

- Provide visibility and control of parameters which control the operation of the NOS/VE system (e.g., scheduling of various classes of jobs, secure or unattended mode of operation, etc.).

- Report existing or pending hardware and software problems (excessive or unrecovered memory/disk/tape errors, free disk space nearly exhausted, power failure, etc.) and allow operator intervention if necessary.

- Allow System Operators to communicate with NOS/VE jobs and terminals according to various selection criteria such as a particular job or terminal, all jobs or terminals, a predefined group of jobs or terminals, etc.

- Allow authorized NOS/VE user jobs to communicate with System Operators.

- Communication of information needed for system operation to the appropriate System Operator(s) will be independent of the operator configuration.

- Provide an interface that allows the following parts of the NOS/VE system to communicate with System Operators:

  On line diagnostics.
  Peripheral equipment managers.
  System utilities.
  Troubleshooting utilities (locate lost job, determine cause of job failure, etc.).

- Provide dynamic display information to system analysts for system problems which cannot easily be analyzed with static information such as dumps, traces, etc.

3.13.4 ASSUMPTIONS AND CONSTRAINTS

The Operator Facility design is based on the following assumptions and constraints:

- NOS/VE must provide the same general operator functions as NOS/170 and NOS/BE but the emphasis in NOS/VE will be on having much less required operator intervention than on NOS/170 or NOS/BE.

- The operator interface for NOS/VE R1 will be through normal interactive terminals on C170 (NOS/170 and NOS/BE). Operation of NOS/VE through the C170 CC545 via the K display on NOS/170 or the L display on NOS/BE will be provided by simulating a terminal interface to NOS/VE.

- The 752 console will be supported through the Two Port Mux in NOS/VE R2.

- The CC545 will be supported directly by NOS/VE native mode in R2 or a later release.

3.13.5 DESIGN APPROACH

The NOS/VE Operator Facility design is based on the following guidelines:

a. The following types of consoles will be supported:

   - Standard System Console (752 Terminal on the Two Port Mux)
   - NOS/VE Interactive Terminals (connected to C170 initially and directly connected to C180 later)

- CC545 via C170 in NOS/VE R1 and driven directly in NOS180 R2 or a later release.

The NOS/VE Operator Facility will support the full ASCII character set. Displays will be normalized to accomodate device dependent characteristics so that they will be available in some form on all types of consoles or terminals which are supported by the NOS/VE Operator Facility.

b. The Operator Facility must allow (but not require) operator functions to be distributed to a set of Operator Consoles (i.e., NOS/VE will not be restricted to a single centralized operator console).

c. All NOS/VE System Operator consoles will be handled as normal NOS/VE interactive users. All System Operators must be valid NOS/VE users and must "login" to the NOS/VE system. Special privileges granted to System Operators will be determined by the location (hardwired terminal address) of the Operator Console and the privileges associated with the particular "Operator User".

d. The same basic interface must be presented to System Operators independent of the type of console being used (within the constraints of the console itself).

e. Displays will be optimized for consoles or terminals which have display screens but terminals without screens will be accomodated. Console or terminal characteristics such as screen size will be parameterized at execution time by the Operator Facility.

f. All system operator command and display requests must be permanently logged in the NOS/VE System Log and in the Job Log of the NOS/VE interactive job of the Operator User.

g. The Operator Facility must allow System Operators to have a dialogue with a NOS/VE job or terminal.

h. The Operator Facility must support an "Unattended Mode" of operation in which situations that would normally require operator intervention, are handled automatically without operator action. The installation will be able to specify what action the system should take if the system is in "Unattended Mode" in situations where operator intervention is normally required.

i. A dynamic display capability must be provided for appropriate types of consoles so that display information is

automatically updated on a periodic basis. Dynamic updating of displays will be provided only for those displays for which it is appropriate.

j. The Operator Facility must be extendable so that standard commands and displays can be added with a minimal amount of special effort (i.e., it must be possible to develop displays and commands as user level programs). This will also allow installations to easily add new commands and displays in order to tailor the operator interface to their particular needs.

k. The external interface to the System Operator like the NOS/VE interface to the normal user must be "human engineered" in order to make NOS/VE easy to operate. Some of the specific things that must be done are:

- Command syntax, display formats, naming conventions, etc. must be consistent.
- Operator commands and display requests must allow abbreviated forms and convenient defaults in order to minimize operator typing.
- Parameter errors must be completely diagnosed and reported with meaningful diagnostic messages.
- Prompting must be provided where appropriate to assist the operator in making correct entries.
- On line command and display dictionaries must be provided.
- Command verbs, parameter names, etc. must be as "English like" and as meaningful as possible.

l. The NOS/VE Operator Facility will provide a normal/abnormal status to System Operators to notify them that the NOS/VE system is running properly or has crashed or hung. NOS/VE Monitor will cooperate with C170 DSD to provide this information for the C170 operator. NOS/VE Monitor will cooperate with the MCU to provide this information for the NOS/VE operator on the Two Port Mux Operator Consoles.

m. NOS/VE Deadstart will not use the Operator Facility to communicate with the operator during the deadstart process. Communication with the NOS/VE operator during deadstart will be through the C170 utility which performs the initial loading of the NOS/VE system to C180 memory. It is expected

that operator communication during NOS/VE deadstart will be minimal and that it will take place prior to the start of the actual deadstart process.

n. NOS/VE On line Diagnostics must be able to share an Operator Console with normal NOS/VE operator commands and displays. In particular, a CE must be able to use an Operator Console to initiate and control On Line Diagnostics which run in the background while the Operator Console is being used for normal system operation. On Line Diagnostics which share the Operator Console with the normal NOS/VE operator commands and displays must use the system interfaces provided by the Operator Facility.

The following approach will be followed for the implementation of the NOS/VE Operator Facility:

a. The NOS/VE operator interface will be distributed rather than centralized (i.e., a number of concurrent operator consoles with different types of privileges will be supported). The installation will be able to assign groups of operator privileges to each Operator User in order to partition operator functions among a number of different Operator Consoles.

b. A variety of consoles will be supported (both normal interactive terminals and special operator consoles) but all consoles will interface to the NOS/VE Operator Facility as normal NOS/VE interactive terminals. Differences between normal NOS/VE interactive terminals and special operator consoles will be compensated for as close to the console as possible (i.e., the CC545 on the C170 will simulate a C170 interactive terminal, the 752 on the Two Port Mux will be made to look like a NOS/VE interactive terminal by NOS/VE Physical I/O).

c. All operator commands and displays will be processed by normal NOS/VE interactive jobs which are granted special "System Operator" privileges through the normal NOS/VE validation process.

d. The NOS/VE Operator Facility will provide an environment in which the Command/Display processors function. This environment will include the following interfaces:

- NOS/VE interactive job created by "Operator User" "login" to NOS/VE.

- Command input through the SCL command interface via GETs

from the interactive INPUT file.

- System Interface Procedures which can be called by privileged users to obtain information from the system and to exercise control over things in the system.
- System Job Interface Procedures which allow information to be transferred to/from NOS/VE System Jobs.
- Display Building Procedures which structure information that will subsequently be displayed at the console.
- A Display Generator that will format structured display information according to console characteristics and send it to the console with PUTs to an interactive output file.

e. The Operator Console display screen will be partitioned by the Display Generator into the following sections:

- A header which contains general information such as system version, time, date, etc.
- An action message area which contains asynchronous requests from the system for operator intervention.
- A display area which contains normal system displays.
- A command response area which provides accept/reject responses to operator command and display requests.

f. The Operator Facility provides the environment for the operator interface. Use of this environment to provide the actual operator interface is part of each area of the NOS/VE system (i.e., Physical I/O will provide commands and displays to control on line peripheral equipment, etc.). The Operator Facility area will implement the display and command processing environment and those commands and displays which do not fall into a particular system area such as a general system status display, display and command dictionary, etc.

3.13.6 NOS/170 DIFFERENCES

The NOS/VE Operator Facility will differ from it's NOS/170 counterpart (DSD) in the following ways:

a. The NOS/VE Operator Facility will support multiple operator

consoles which allow system operator functions to be distributed among a set of consoles rather than just a single centralized console as on NOS/170.

b. A variety of console types will be supported by the NOS/VE Operator Facility (752 on the Two Port Mux, NOS/VE Interactive Terminals, etc.) rather than only a single special purpose console (CC545) as on NOS/170.

c. The NOS/VE Operator Facility will utilize the full ASCII character set (if the console allows it) rather than just the Display Code subset supported by the CC545 on NOS/170.

d. All operator command and display processing will be through normal NOS/VE interactive jobs rather than by a dedicated PPU driverr. This means that system operators must login, logout, etc. to the NOS/VE system.

e. SCL syntax will be used for NOS/VE operator commands and display requests rather than NOS/170 DSD syntax.

f. Automatic syntax recognition (command fill in) as provided by NOS/170 will not be provided for NOS/VE operator commands and display requests - optional abbreviations will be used instead.

g. Small, medium and large display characters will not be supported by the NOS/VE Operator Facility.

h. NOS/VE operator command and display verbs will in general be different than those on NOS/170.

i. NOS/VE display organization, content and formats will be different than on NOS/170.

j. Display intensification and phasing to attract operator attention as on NOS/170 will not be provided by the NOS/VE Operator Facility. Highlighting portions of a display in order to attract the attention of the operator will be supported if the console being used supports it. The method used to provide highlighting (blinking, character inversion, etc.) will depend on the console characteristics. If a console supports more than one highlighting method, only one of the methods will be used.

3.14 CPU MANAGEMENT

3.14.1 GENERAL RESPONSIBILITY

CPU management functions include task dispatching, signal routing, interval timer management, error processing, PP request queuing, real memory management, page faults, and interrupt processing. These functions are the basic interface to the hardware and represent the most primitive layer of NOS/VE.

3.14.1.1 Glossary

Page Frame Table - A page frame table entry exists for every page frame in the system. Each entry contains page status, usage data, and pointers for queuing to other frame table entries. A page frame is always on one of the following queues: free, available, available modified, shared, job (working set) or wired.

Page Table - The hardware page table.

3.14.2 DESIGN OBJECTIVES

Integrity

A CPU monitor operates at the most privileged level within the system and has access to system wide tables making the integrity of CPU monitor very critical. The following guidelines improve upon CPU monitor integrity:

- Whenever practical, move functions out into system tasks or task services. Keeping CPU monitor as basic as possible.
- Minimize the sharing of memory and tables between CPU monitor and jobs.
- Serialize as much of CPU monitor as possible, i.e., only one processor within CPU monitor at a time.
- Do not build in automatic retry of hardware detected malfunctions within critical system modules. Return status to higher level modules who execute recovery functions.

- Build in trace facilities (e.g.,keypoints, last N interrupts stack) which will assist in verifying the correct operaton of CPU monitor.

Efficiency

Since all inter-job communication, basic I/O processor assignment, real memory management, and interrupt handling is performed by CPU monitor, efficiency must be a design objective. A simple design which uses the hardware in a straightforward manner helps efficiency. Additional efficiency guidelines include:

- Have traps enabled when executing CPU monitor allowing I/O operations and inter-processor communication to be handled in a responsive manner.

- Consider implementing critical portions of CPU monitor in assembly language. The initial design of table and module interfaces must be done in PASCAL.

- Allow the normal case to be handled by resident code and low overhead interfaces (branch) and the abnormal case (table overflow, errors) to be handled by non-resident code and more flexible interfaces (signal).

- Design lowest level modules to be machine specific.

- Use a simple approach when accounting for usage of basic resources (shared pages, CPU time, PPU time, channel time, etc.).

Resource Utilization

CPU monitor is responsible for allocation of the most basic preemptive resources; processors, channels and real memory. A reasonable compromise between giving top services to highest priority tasks and utilizing resources at maximum efficiency must be an objective. In some cases, dedicating or reserving a resource for future requests may improve utilization. Examples include: maintaining a pool of free pages, dedicating PPU(s) to disk I/O, maintaining task id's constant for a job's life, maintaining ASID's constant for a job's life, etc.

3.14.3 ASSUMPTIONS AND CONSTRAINTS

- The initial releases of NOS/VE use disk as a paging medium, i.e., no unique paging device.
- A minimum central memory configuraton is 1-megabyte.

3.14.4 DESIGN APPROACH

CPU monitor executes in monitor mode, is wired down, has traps enabled, interrupts disabled, and is entered via an interrupt or trap. When executing, CPU monitor will have the following monitor conditions enabled (MCR bits set) power warning, exchange request, external interrupt and system interval timer. The remaining monitor conditions are disabled, and if they occur, will cause the CPU to halt. All arithmetic conditions are disabled (UCR bits 07-15 cleared) when executing in CPU monitor.

On interrupt entry, CPU monitor has access to which task had been executing. Based on the type of interrupt, parameter data will be read from the tables which describe the task (exchange package, segment descriptor table, processor state registers).

Parameters from a task to system monitor via a system call are contained in the tasks registers. The parameters are organized into a request block which contains a unique code for each request type and is used by CPU monitor for routing purposes.

3.14.4.1 Processor Assignment

Processor assignment is performed by the dispatcher. Only tasks which have a status of ready are candidates for assignment. The order for assignment is determined by priority: highest priority first, with a round robin assignment used across tasks of the same priority.

3.14.4.2 Exception Conditions

Exception conditions not directly related to the executing task and which do not preclude further processing and handled as follows:

Exchange Interrupts - This indicates a PPU has issues an MEJ instruction. CYBER 180 CPU monitor gives control to CYBER 170

CPU monitor.

Power Warning - CPU monitor attempts to selectively dump real memory, display a "POWER FAILURE IMMINENT" on the system console and, if possible, to the terminal users. The condition will be monitored for 15 seconds by the MCU. If the fault was transient, the system is restarted.

External Interrupt - This indicates completion of an I/O operation. These are recognized immediately and cause queued requests to be submitted for processing.

System Interval Timer - The system interval timer represents a system wide real time clock which is used for:

- Monitoring the CPU time used by a task.
- Controlling the execution quantum for a task.
- Allowing a task to relinquish the CPU for an element time period.
- Maintaining the date and time of day.
- Detection of requests which have exceeded wait (time) limits.

3.14.4.3 Paging

The general paging approach is to maintain a pool of free pages for satisfying page faults and page assign requests. Whenever the free pool drops below a threshold, the replacement algorithm is aplied to the existing assigned pages, to replenish the free pool. Assigned shared pages are maintained on a system basis according to frequency of use. The pages which have not been used for the longest time period become candidates for reassignment.

The NOS/VE replacement algorithm assumes that most programs will tend to cluster references to certain pages within a time interval. This property is commonly known as locality with the set of pages referred to as working set. Occasionally during the execution of a program, a change of locality will occur, e.g., between different phases of a compiler, or the number of pages within the reference cluster may increase or decrease. These changes will be recognized and adpated for by the algorithm with the objective being to minimize the number of page faults.

In general, the algorithm adds pages to the working set on demand (page fault) and subtracts pages by periodically aging them and removing from the working set all pages whose age increment exceeds a ceiling value. Newly added pages must be part of the working set for a minimum time equal to a floor value. New pages are always assigned from the available queue(s) and removed pages are always added to the available queue(s).

The frequency of page faults within a job is used as an adaptive parameter to control the replacement algorithm. When the page fault frequency increases, the values for ceiling and floor decrease resulting in pages being removed from the working set earlier. When the page fault frequency decreases, the value(s) for ceiling and floor increases resulting in pages staying in the working set longer. Whenever the free pool drops below a threshold, the job working sets will be aged regardless of the page fault frequency rate.

The algorithm is applied to all job working sets. Pages which are shared between jobs are aged on a global basis.

3.15 NOS/VE MAINTENANCE SERVICES (NOSMS)

3.15.1 GENERAL RESPONSIBILITY

NOS/VE Maintenance Services is a collection of NOS/VE software which supports hardware test/diagnosis, remedial maintenance, and preventive maintenance of non-critical system components concurrent with customer operation. It enables repair of non-critical system components to be deferred to a suitable maintenance period.

3.15.2 GLOSSARY

CEM - Configuration Environment Monitor. A peripheral device on the MCH and having the ability to detect environmental conditions (dew point, MG status, etc.) and control power on/power off to C180 equipment.

CPU Monitor - The portion of the operating system most directly related to the hardware environment. One of CPU monitor's function is to provide the interrupt handling for hardware faults.

Critical Component - A hardware component of a NOS/VE system without which the OS cannot function and off-line maintenance or Deadstart/Recovery must be performed. (Reference 1.3.2.)

CTI - Common Test and Initialization. This is a process whose function is to bring the C180 hardware to a known state prior to NOS/VE Deadstart/Recovery execution. It includes operator dialog, firmware loads/dumps, hardware validation, mainframe attribute determination and OS bootstrap load.

Deadstart/Recovery - (Also NOS/VE Deadstart/Recovery) This is the NOS/VE software initialization package. The primary function is to create/recover the NOS/VE environment including:

- permanent files, log files
- I/O queues (spooled files)
- hardware configuration descriptors
- user/system jobs (including local files)

Diagnostic - This is a program (software, firmware or both) which isolates a hardware failure to a service level component.

HPA - Hardware Performance Analyzer. A program which analyzes the Engineering Log and produces reports to aid in preventive and remedial maintenance.

MAC - Maintenance Access Control. A system hardware component which interfaces to the maintenance channel (MCH). It processes the model independent and model dependent (if any) functions received via the MCH.

MALET - Maintenance Application Language for Equipment Testing. A system for hardware test consisting of a language definition and compiler, an executive program and test programs.

MCH - Maintenance Channel. The NOS/VE IOU channel connected to all maintenance access control in a mainframe. It provides maintenance and status functions.

MCR - C180 Monitor Condition Register (Reference MIGDS).

MCU - Maintenance Control Unit. The single PP in a C180 mainframe dedicated to on-line maintenance. The System Maintenance Monitor (OLMM) program is resident in the MCU.

MMR - Monitor Mask Register (Reference MIGDS).

MSL - Maintenance Software Library. An area reserved on system mass storage for on-line/off-line maintenance software storage.

NOSMS - (NOS/VE Maintenance Services). The NOS/VE portion of the overall C180 OLMF.

OLMF - On Line Maintenance Facility. Collection of software and firmware which supports the myriad requirements of concurrent maintenance.

OLMM - On-Line Maintenance Monitor. OLMM is the OS PP program responsible for MCH monitoring/driving. By definition, it resides in the MCU.

Test - A software procedure whose function is to detect hardware errors. (Reference Diagnostic)

VLEX - Virtual Level Executive. On-line VLEX is an interface package which will provide a means for a subset of off-line VLEX tests to run on-line.

3.15.3 DESIGN OBJECTIVES

- NOSMS will collectively be capable of detecting and reacting to all hardware reported errors in the system. The reaction of NOSMS must not compromise system integrity.

- Use reconfiguration and degradation features of the hardware and operating system to defer maintenance to a more cost effective repair period.

- Facilitate continued useful customer work at all degraded levels down to the minimum hardware configuration required by the operating system.

- Ensure that recoverable errors are transparent to the user.

- Maintain a binary format Engineering Log of hardware error and usage data as per ARH2709 (Maintenance Information Logging on NOS/VE).

- For virtual environment maintenance:

  - The primary system for supporting concurrent maintenance of C180 hardware is the 170 operating system for both A170 and virtual environment operation in R1.

  - Allow gradual migration of maintenance software from 170 to 180. While it is planned that NOS/VE will provide a significantly enhanced on-line maintenanc environment, the amount of maintenance software available to take advantage of enhanced features in the R1 timeframe is limited. The NOS/VE hooks for this migration are available in R1.

  - In time critical system emergencies, the 170 system can preempt shared resources. That is, when an environmental warning occurs, the 170 system will be given priority in performance of its shutdown functions; e.g., checkpoint and/or step. NOS/VE requirements for CPU, peripherals, etc. are subordinate.

  - Ensure minimum impact on existing 170 software. The 170 system should not have to greatly change the way it performs error checking and related functions.

Specifically, MTR and 1Mx should not need modifiction to support virtual environment and standalone A170 environment.

3.15.4 ASSUMPTIONS AND CONSTRAINTS

- NOSMS applies only to a single C180 mainframe. Multiple mainframes require multiple Maintenance Services (including dual state operation).
- NOSMS applies only to 180 state hardware (including peripherals). 170 state components in dual state are not maintainable by NOSMS.
- Only single failures are automatically handled by NOSMS. That is, only one hardware component can be automatically tested/diagnosed at a time. Other test/diagnostic sequences may be manually (via CE) activated.
- There must be at least one spare or nonessential PPU for use by on-line maintenance programs (such as MALET).
- Standard PPU/CPU protocol for NOS/VE will be used by maintenance programs.
- The MCH can be dedicated to maintenance for brief periods.
- The MCH will be shared by the performance monitoring facility during normal (non-maintenance) system operation.
- The MCU/MCH are not critical components for normal (non-fault) NOS/VE operation.
- A NOS/VE fault will not cause the MCU to fault.
- Local and remote maintenance access will have the same capabilities.
- The 2 port MUX is driven by NOS/VE and one port is dedicated to maintenance operations (local or remote). The other port is available for use as a system console communication path.
- A communication path to a system or CE operator will be available to OLMM without OS assist.
- On-Line Maintenance is applicable only to non-critical system components. The following treatise deals with the concept of

system critical components:

- CPU
  A C180 CPU will be a critical component if it is the only CPU in a mainframe.

- Peripherals
  Unrecovered errors in C180 central memory will have 1 of 3 possible impacts on system activity.

  1) No interruption need occur (not critical). This is the case where the memory failure has occurred in such a fashion as to not impact NOS/VE critical code/structures. This may be the case when an error occurs in a user assigned page frame.

  2) A system restart must be performed with possible memory reconfiguration (bank interleave select/deselect). This is the case where the failure has occurred in such a fashion as to affect the integrity of critical NOS/VE code/structures. The system will halt.

  3) System goes off-line until memory is repaired. This is the case where the memory failure has become critical and an off-line strategy must be employed. A critical memory failure is one or more of the following:

     a) A path failure including:
        transmission cables
        ports
        fanin/fanout
        secded
        parity generaton/checking

     b) Certain multiple failures in a storage unit which result in insufficient memory capacity.

- Peripherals

  A NOS/VE peripheral RMS media will be a critical component of a NOS/VE system if it contains the only copy of the system library or if it is the only RMS media used for one or more of the following:
  swapping
  paging
  permanent file storage
  spooled files
  temporary files
  etc.

Media failures will be critical only if the failed area contains critical information (such as a critical O/S function page).

In addition to critical media, other components of a peripheral subsystem may also be critical. These include:

1) The RMS drive where a critical media is maintained.

2) The single access controller to which the drive is connected.

3) The IOU channel, barrel, etc. to which the controller is connected.

4) The central memory port to which the IOU is connected.

PPU's need not be critical components if there are spares but they are never diagnosed via on-line maintenance procedures.

3.15.5 DESIGN APPROACH

Hardware fault detection is dispersed throughout the NOS/VE system into such areas as system monitor, OLMM, queue managers, etc. Each area which detects a fault condition is responsible for reporting it to the maintenance control facility (the maintenance job). The reporting mechanism is indirect (via the engineering log buffers) for some faults and direct (via signal) for others.

The decision as to how to react to a fault condition lies with the maintenance job. This will include decisions on what to log, what the logging formats should be, whether reconfiguration is indicated, whether an automatic test/diagnostic program is to be initiated, whether to inform the operator, etc.

For those faults which cause NOS/VE to fail, a backup facility is provided by the OLMM. OLMM has limited capacity for operator communication and automatic system recovery initiation and for passing failure data in each case.

3.15.5.1 Virtual Environment Error Logging

Each state will log error and usage information for each system component it either wholly owns or shares, the key being that each system will log errors it detects and usage it accumulates. This means that mainframe errors will be logged in 170 and 180 system logs.

The primrary advantage of this approach is that no changes need be made in existing/planned logging schemes. NOS/VE logging will not be constrained by A170 system log formats and conventions and can utilize planned compaction/thresholding techniques.

Another advantage is that a clear and immediate path exists for shifting HPA functions from the 170 to the 180. 180 log data will be available in R1 to support mainframe HPA although it need not be utilized because the 170 log will have similar mainframe data. 180 peripheral data is also available in the 180 log at R1 and could be made available to the 170 via a copy/normalizer program or accessed directly via 180 HPA. Further down the migration pat, the HPA emphasis could shift to 180 with copy/normalizer programs to provide the 180 with 170 data.

The primary disadvantage of this approach is that in order to perform HPA for NOS/VE owned peripherals, either a NOS/VE HPA or a 180 to 170 log copy/normalizer must be available. Further, any such copy/normalizer must be able to equate equipment in 180 state to equipment in 170 state.

3.15.5.2 NOS/VE Error Processing

The OLMM will periodically examine the summary status bit of the IOU status summary register and upon finding the bit on, will examine the status registers of mainframe components to determine specifically what the hardware error condition is.

Normally, OLMM will pass error information to the maintenance job for further action but for hardware errors causing system failures, the OLMM can activate automatic system recovery with reconfiguration or leave hardware error data intact for all line analysis.

Actions caused and/or taken by the OLMM will include the following:

- Log the error (including thresholds)

- Clear the error indicators
- Automatic system restart
- Automatic reconfiguration
- Test/diagnostic activation
- Perform a checkpoint
- Perform a step
- Display system down indication

In addition, the NOS/VE monitor will process CPU interrupt conditions passing error information to the Maintenance Job.

3.15.5.3 Virtual Environment Error Processing

In the virtual environment, each system's mainframe error detection and reaction scheme functions as close to standalone mode as practical. There are two areas of interaction which must be examined further.

3.15.5.3.1 COORDINATION OF MCH USAGE

Usage of the MCH and various hardware status registers accessible via the MAC interface must be coordinated between the two systems. The usage of the MCH will be distributed among the PP monitor and 1Mx for 170 and the CLMM and PMF for 180. These four independent processes must synchronize their actions in an efficient manner that does not tie dual systems together via intersystem memory references or other system dependent methods.

The MCH itself is used for this synchronization via combinations of channel flag and channel full states. This coordination method could be used in standalone A170 environments to minimize the amount of processing differences when in the virtual environment.

In addition to MCH usage coordination, the two systems must coordinate clearing of hardware "error logs" or status registers to ensure that both can detect an error condition before allowing a new condition to be recorded. For example: the corrected error log for CM cannot be written when there is a valid entry indicated therein. This coordination between 1Mx and CLMM is performed over the MCH with CLMM actually peforming the MCH write of the applicable maintenance register.

3.15.5.3.2 COORDINATION OF OTHER SYSTEM ACTIONS

For some of the hardware error conditions the 170 and 180 systems must coordinate more than simple logging and clearing of hardware status registers. In particular, coordination of system

checkpoint and step mode must be considered as well as planning for recognition and subsequent reaction to an opposite system error condition.

A key issue in the coordination of system checkpoin are system step mode is how to allocate shared resources (CPU and peripheral devices) in time critical situations such as log and short power warnings where it is conceivable that there is inadequate time for both systems to respond to the emergency. Again, the emphasis should be placed on 170 recoverability in early virtual environment systems as the NAM/IAF, operator facility, unit record, and bulk of file storage is handled by 170. These factors place a requirement on the virtual environment design to be able to cause CPU and other shared resources to be deciated to one system or the other until a specified set of actions have been accomplished.

It is adequate to treat operator initiated 170 checkpoint and step mode in a similar fashion to time critical checkpoint and step mode but it may be more desirable to force the operator to manually checkpoint or step the 180 system first.

Each system must also be capable of recognizing and rejecting an opposite system failure whether hardware or software generated. In particular, 180 recognition of a 170 failure should at least allow a step condition to occur and at best a 180 checkpoint. 170 should at least be able to notify the operator of 180 failures and at best be able to support an automatic 180 recovery/deadstart.

3.15.6 NOS/170 DIFFERENCES

The primary difference between the on-line maintnenance facilities of NOS/170 and NOS/VE is that NOS/VE dedicates a system job to provide a centralized maintenance reporting and controlling point. This enables more automatic programmed reaction to hardware fault conditions. It also provides a simpler interface between the maintenance software defined by CDED and the related facilities provided by NOS/VE (defined by CPD).

3.16 NOS/VE DEADSTART/RECOVERY

3.16.1 GENERAL RESPONSIBILITY

NOS/VE Deadstart/Recovery (subsequently referred to as Deadstart) is the function which activates the NOS/VE operating system on hardware which is in a known state. The deadstart process leaves the NOS/VE system in a state which is ready to execute customer workloads.

Deadstart includes the recovery of permanent file bases, system log files, I/O queues, hardware configuration information and user/system jobs (including temporary files).

3.16.2 GLOSSARY

Basic O/S - The smallest subset of NOS/VE which will support the deadstart job.

CMSE (180) - Common Maintenance Software Executive. The off-line 180 diagnostic system.

CTI (180) - Common Test and Initialization. This CDED product provides a common front end to deadstart for all 180 systems. Reference 1.4.1.

Critical Error Log - An area on the RMS deadstart device reserved for use by the off-line maintenance system, CMSE (Common Maintenance Software Executive) for logging system hardware error information.

Deadstart Device - CTI can operate from tape or RMS as will NOS/VE deadstart. Hence, the term deadstart device will apply to either class of device. The RMS deadstart device will be the system library and MSL device.

Deadstart Job - The first NOS/VE system job to execute at each deadstart/recovery. This job is functionally a part of NOS/VE deadstart.

Error Interface - (EI) EI provides the C180 interrupt handling capabilities for NOS/170 when running on the C180 hardware.

Executing Job - A job which is not swapped out or in the process of being swapped. At least a minimum of the job's address space (working set) is CM resident.

HDT - (Hardware Descriptor Table) This table is built by CTI and contains information describing mainframe attributes including:

- CM size
- Processor counts and types
- Barrel count
- Channel numbers
- Processor and maintenance register values
- PP on/off status

MCU - (Maintenance Control Unit) The single PP in a C180 mainframe dedicated to on-line maintenance and related functions.

MSL - (Maintenance Software Library) A library of maintenance programs residing on the RMS deadstart device. MSL programs consist of on-line, off-line, and common (either on-line or off-line) programs.

O/S Boot - A program on the CTI deadstart device or the NOS/170 system library (dual state) which is loaded into a PPU and given the control of the NOS/VE deadstart process. Due to CTI considerations, this program is limited in size to one sector on the RMS deadstart device.

Swapped Job - A job which is in a state such that all information needed to continue its execution is on RMS. Further, the job is not subject to execution until a swap-in process occurs. Characteristics of a swapped file include:

- All non-shared non-system segments of the address space are RMS resident.
- All system tables relating to the job are RMS resident or backed up on RMS. These include: file tables, equipment assignment information, job defined queues, SCL defined environment, accounting information, etc.
- Each task's exchange package and segment table are RMS resident.

System Checkpoint File - A system checkpoint file is an RMS file which contains information sufficient to reestablish

the NOS/VE operating environment to the point when the checkpoint was taken. This includes recovery of all queued I/O files, permanent and temporary files, and all jobs known to the system at checkpoint time. The act of taking a system checkpoint may be operator initiated (as in the case of termination of a scheduled NOS/VE operation prior to off-line maintenance) or it may be system initiated (as in the case of environmental condition warning). In any case, a system checkpoint action is followed by a system halt. System restart invalidates the current checkpoint file.

System Job - A job which is always "active" in the NOS/VE system. System jobs are permanently installed on the system set. System jobs peform such functions as:

link facility
operator communication
on-line maintenance

System Set - A set of one or more logically non-removable RMS devices which include:

the primary system library device
the MSL device
the RMS deadstart device
at least one permanent file device
at least one temporary file device
at least one queue file device

System Source - At deadstart time the location of the system library (system source) can be one of the following:

- local tape
- local mass storage
- linked mainframe RMS including the dual state link

3.16.3 DESIGN OBJECTIVES

- Provide several levels of deadstart/recovery:

1. Install (no recovery)
2. Recover permanent file base
3. Level 2 plus I/O queues
4. Level 3 plus swapped jobs
5. Level 4 plus executing jobs
6. Recover from a system checkpoint file

- Support a "no operator" mode of deadstart/recovery.

- Keep the unique PP and CP code requirements to a minimum (e.g., utilize as much standard system code as possible).

- Minimize dependencies on factors such as system source and system set device types.

- Support automatic deadstart/recovery with logical/physical reconfiguration including:

  Mainframe: Add/delete CPU
  Change memory interleave
  Change page size
  Add/delete barrels and individual PPU's
  Add/delete channels

  Peripherals: Add/delete controllers
  Add/delete devices

3.16.4 ASSUMPTIONS AND CONSTRAINTS

- NOS/VE deadstart must function in dual state and stand alone environments.

- Virtual environment operation will involve a dynamic deadstart of NOS/VE activated from the 170 state. NOS/170 will not be halted during the 180 deadstart process.

- Recovery of executing NOS/VE jobs requires a valid central memory dump on the RMS deadstart device. This will be provided by CTI or by "dual state initialization".

- The basic O/S and deadstart job environment is non-paged and must operate in 1MB of memory.

- Wherever feasible, NOS/VE system functions must provide an initialization state which will be responsible for establishing their respective operating environments. E.g., each function will build its own tables in central memory (or RMS), initialize its own firmware and drivers (where applicable), and establish its own settings of hardware registers, etc.

- Software activated deadstart/recovery does not include master clear. This hardware limitation can preclude automatic

recovery from certain faults occurring in the IOU such as CM read errors, etc.

- CTI 180 will provide firmware loads for processors.
- The CTI 180 deadstart device and console drivers are suitable for use as drivers by the O/S boot process.

3.16.5 DESIGN APPROACH

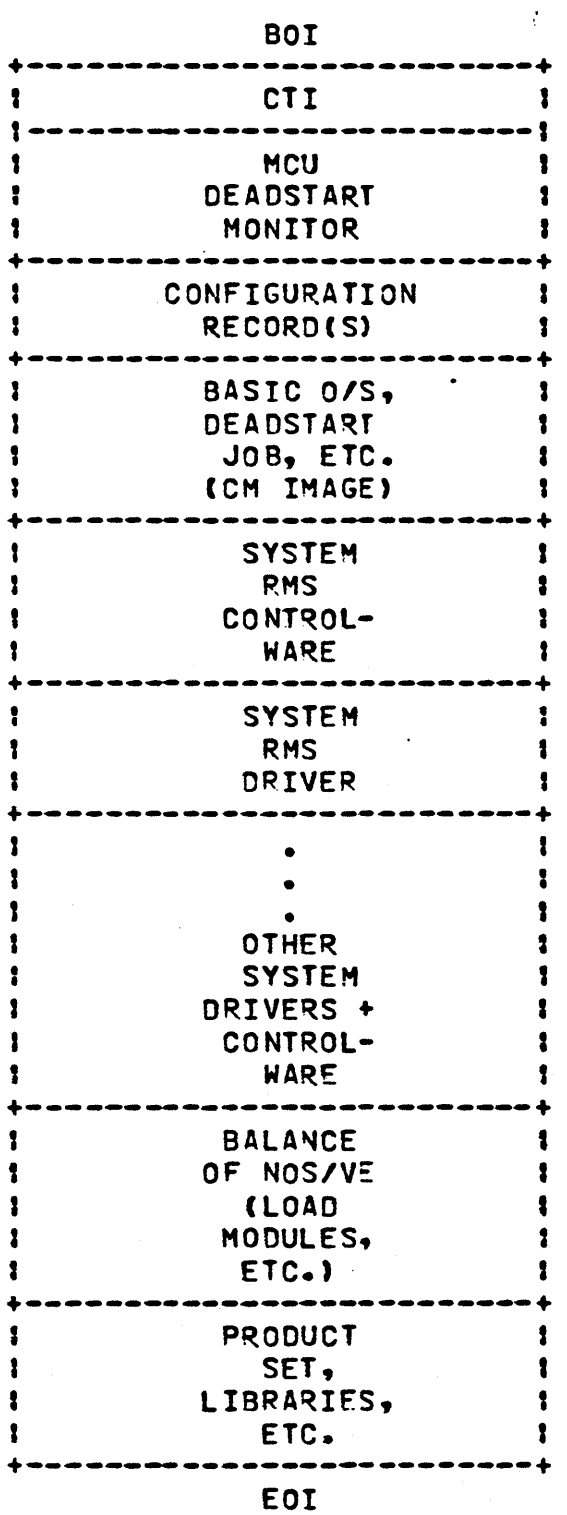
The design approach is arrived at chiefly by comparisons of existing CDC deadstart techniques for SCOPE 2, NOS/BE and NOS. SCOPE 2 and NOS/BE have extremely large monolithic deadstart programs which duplicate many functions performed by the systems they are deadstarting (examples include: system drivers, RMS allocation, operator communication, command language processing, etc.). NOS deadstart, however, achieves a minimum OS capability and then builds on this through the activation of special jobs (control points) to initialize such things as the magnetic tape subsystem or transaction subsystem or time sharing facility. This is the process which is felt to be advantageous and hence NOS/VE deadstart/recovery is loosley patterned after NOS deadstart/recovery.

NOS/VE deadstart is front ended by CTI 180 for stand alone mode and by a dual state initialization process in dual state environments. In either case, parameters will be passed to the NOS/VE deadstart in PPU memory. These parameters will include mainframe attributes such as processor kinds and numbers, memory size, and barrel and channel specification. The parameters will also include the deadstart panel settings when front ended by CTI or a simulated deadstart panel when front ended by dual state initialization. The deadstart panel settings will include the level of recovery to be performed. (Ref 1.3.1)

A key factor in the design is that the recovery of central memory structures is achieved by utilizing RMS for saving and retrieving the structures. Saving CM is performed by the dual state dump facility or the CTI disk dump facility or by the checkpoint facility. A checkpoint facility is treated as a special case of the general system dump files created by the dual state dump facility and the CTI disk dump facility.

The current deadstart design approach also impacts the structuring of the deadstart media or file (system source). The following diagram shows the order in which information is processed from the system source.

DEADSTART FILE FORMAT



3.16.6 NOS/170 DIFFERENCES

The primary external difference between NOS/170 deadstart/recovery and NOS/VE deadstart/recovery is that the normal deadstart/recovery operation for NOS/VE will involve no operator dialogue.

Internally, the NOS/VE system will be more keyed to the concept of an initialization state. This will enable fewer interdependencies between 180 deadstart/recovery and the NOS/VE operating system.

3.17 PROGRAM MEASUREMENT AND ANALYSIS

3.17.1 GENERAL RESPONSIBILITY

The Program Measurement and Analysis Facility (PMAF) shall assist users in the efficient development of effective programs. PMAF provides aid in the analysis of program execution time distribution and memory reference distribution.

3.17.2 GLOSSARY

Block - A contiguous sequence of instructions or data storage locations. The PMAF will partition a program into blocks and perform measurements based on these units.

Inter-block Reference String - A chronological accumulation of elements each of which depicts the occurrence of a reference from one block to another.

Locality - A set of program blocks which reference only other blocks within the set over a significant interval of execution time.

Locality of Reference - The degree to which a program's execution is characterized by infrequent transitions from one (relatively small) locality to another.

Program Profile - The distribution of a program's execution time over program blocks.

3.17.3 DESIGN OBJECTIVES

- The Program Measurement and Analysis Facility is intended to provide a tool to facilitate efficient development of efficient and readable programs. Program profile reporting allows the programmer to efficiently reduce overall execution time by optimizing (only) the most frequently execution portions of the program. Automatic program restructuring improves program locality of reference -- thereby reducing memory requirements -- without requiring source code restructuring which could cause diminished readability.

- The program restructure capability should be easily applied to a large class of programs to obtain significant improvement in locality of reference. Specifically, the program restructuring facility should be effective for programs written in FORTRAN, PASCAL-X, and COBOL (in order of importance).

- Implementation of program instrumentation must be accomplished in a manner which limits the artifact induced in the measured program (in terms of both CPU demand and memory demand) to reasonable levels. Ideally, the artifact would itself be measured and factored out of any analysis.

3.17.4 ASSUMPTIONS AND CONSTRAINTS

- All program measurements communicated to the programmer via legible reports must be reported relative to blocks defined in the source program rather than units reflecting the underlying machine structure.

- Source code manipulation shall not be utilized as the primary method for program restructure. The PMAF may ultimately be enhanced to support this type of restructure but the basic capability must not rely on this approach.

- In order to obtain greatest effectiveness in program restructure, program blocks must be relatively small compared to page size -- ideally less than one third of page size. In order to obtain maximum effectiveness for the target source languages, the following compiler constraints must be imposed. For PASCAL-X, each procedure should cause generation of a separate code section. For COBOL, the code for each paragraph which is executed only via PERFORM should be contained in a unique code section. The requirement for COBOL is extremely important, for otherwise entire COBOL programs will appear as one monolithic object text which cannot be restructured by automated means.

- The same technique shall be employed for obtaining program profile and program locality measurements.

- In order for the PMAF to be effective, the program must be modular in nature. It is assumed that a large percentage of programs executed on NOS/VE will fall into this class.

- The design approach taken will be extensible to large monolithic programs subject to the following constraints. The

instrumentation methodology must be different since measurements must be made relative to blocks known only to the compiler. Program restructure must be accomplished manually since source code manipulation is required.

- The PMAF will restructure a program such that the program's locality is optimal when executed with a specific set of input data. The degree to which this restructuring improves locality when the program is executed using different input data is limited by the dependency of the program's reference behavior upon the program's input data. Previous research has indicated that a large class of interesting programs exhibit data independent referencing behavior.

- The PMAF will address only instruction block localities. No attempt will be made to determine data storage localities. Previous research supports the hypothesis that such an approach nets significant results.

- Each code section contained in the object text generated by a source language translator is assumed to be entered only via the CALL INDIRECT instruction (with both Code Base Pointer and Binding Section Pointer specified).

- The library generator is assumed to possess a capability for handling multiple code sections within a module separately.

- It is assumed that although the PMAF program restructuring capability will be oriented towards a specific paging policy (the damped working set policy), it will not be sensitive to details of implementation of that policy.

3.17.5 DESIGN APPROACH

Programs are instrumented and executed using typical input data in order to obtain measurements of program characteristics. The instrumentation process is independent of the source language in which the program is written; conceptually, it occurs as part of the program loading process. However, the granularity of the measurements taken is limited by the nature of the object text produced by the source language translator. The basic unit of measurement (a block) is a code section.

The result of execution of the instrumented program is an inter-block reference string. There is an element in the string for every transition from one block (code section) to another -- in chronological order. Each element contains an identifier for

the code section transferred to, plus the time of occurrence of the transfer. The inter-block reference string may be analyzed to determine two interesting program characteristics. A program profile can be generated which identifies the most time-consuming blocks of the program. Also, the collections of blocks which constitute localities may be defined.

Once the program's localities have been determined, the blocks may be restructured from their natural order into an order which makes the program's reference behavior conform more closely to the type of behavior expected by the system's memory management policy. Such a reordering should minimize the memory demand generated from each locality by minimizing the page fault frequency and reducing the program's average working set size.

Ultimately the inter-block reference string gathered by the PMAF may prove valuable for other purposes, e.g., debugging, program tracing and determining code coverage during testing.

3.17.6 NOS/170 DIFFERENCES

NOS/170 has no comparable capability for measuring and analyzing program memory reference distribution.

NOS/170 makes available to users an unsupported capability for program profiling called SMP. SMP differs from PMAF in that it measures execution time distribution via periodic sampling of the P-register. SMP measurements are reported to the user in terms of machine addresses instead of source language symbolic names. SMP offers the capability of restricting the range over which measurements are reported to an arbitrarily small subset of a program. (In the case of PMAF, the subset to be measured would need to be isolated on a separate file and could be no smaller than a module.)

PART II

DESIGN ANALYSIS

TABLE OF CONTENTS

TBF = to be furnished

1.0 INTRODUCTION

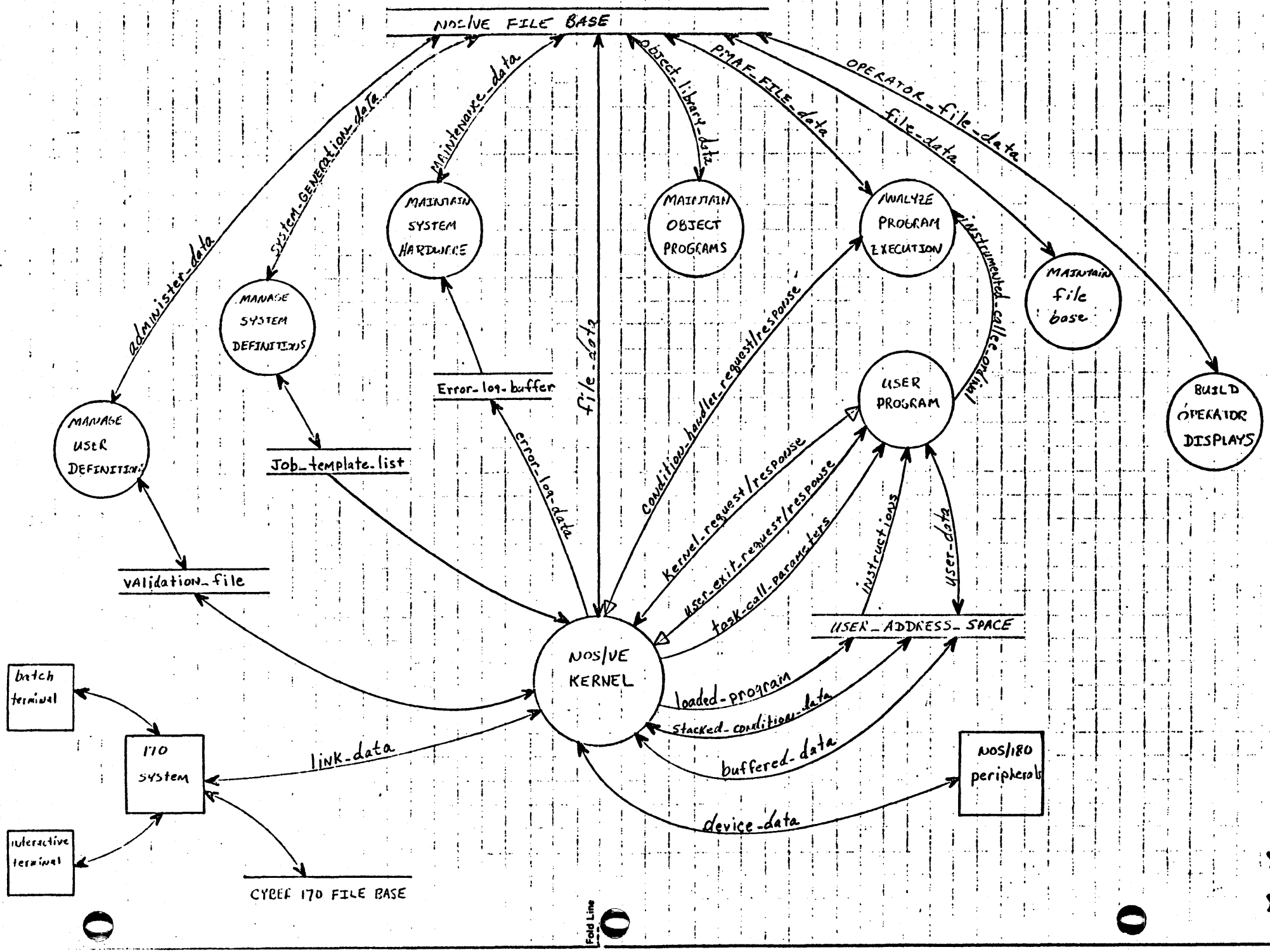
1.0 INTRODUCTION

This part of the NOS/VE Design Specification provides an analysis of the data processed by the operating system and the processes that operate on that data. The information is presented using data flow diagrams and a data dictionary. The data dictionary includes definitions of the data and process descriptions for the lowest level processes.

2.0 NOS/VE CONTEXT

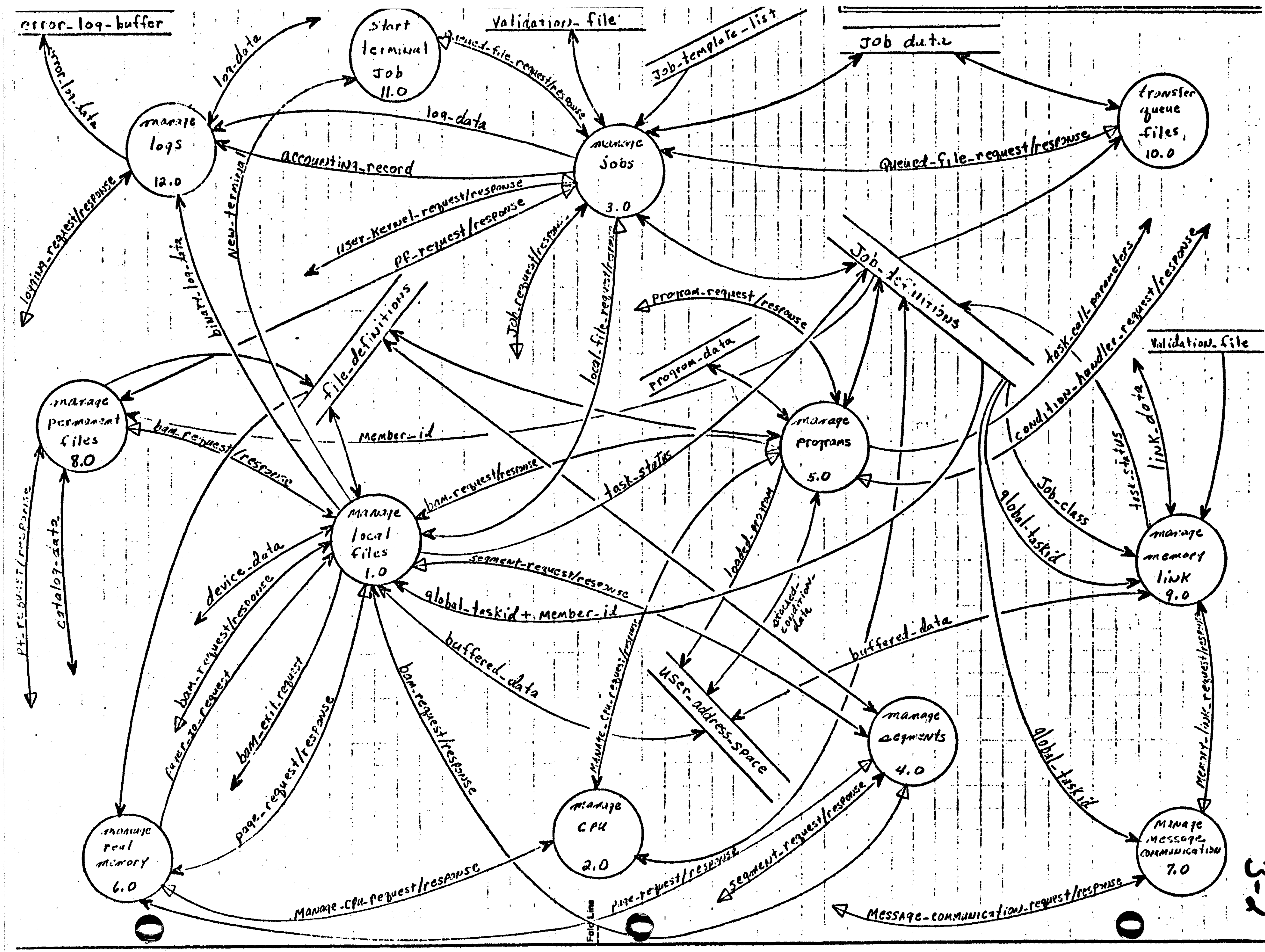
2.0 NOS/VE CONTEXT

NOS/VE CONTEXT DIAGRAM 2/23/77



3. NOS/VE KERNEL

3.0 NOS/VE KERNEL



error_log_buffer
Start terminal Job 11.0
Validation_file
Job_template_list
Job data
transfer queue files 10.0
manage logs 12.0
manage Jobs 3.0
log_data
error_log_data
log_data
accounting_record
queued_file_request/response
loggin_request/response
user_kernel_request/response
pf_request/response
Job_request/response
local_file_request/response
binary_log_data
New_terminal
Job_definitions
file_definitions
program_request/response
program_data
task_call_parameters
condition_handler_request/response
Validation_file
manage permanent files 8.0
Member_id
bam_request/response
manage programs 5.0
task_status
link_data
Job_class
global_taskid
manage local files 1.0
bam_request/response
manage memory link 9.0
device_data
segment_request/response
global_taskid + Member_id
loaded_program
attached_condition_data
catalog_data
pf_request/response
bam_request/response
bam_exit_request
buffered_data
buffered_data
page_request/response
bam_request/response
Manage_cpu_request/response
user_address_space
manage segments 4.0
global_taskid
memory_link_request/response
manage real memory 6.0
manage CPU 2.0
Manage_cpu_request/response
page_request/response
segment_request/response
Manage Message communication 7.0
Message_communication_request/response
Fold Line
3-2

```
BUFFERED_DATA : FLOW
  = ( USER_DATA | SB_DATA );
DEVICE_DATA : FLOW
  = ( DISK_DATA | TAPE_DATA | MCU_DATA );
FILE_DATA : FLOW
  = ( CATALOG_DATA | JOB_DATA | LOG_DATA | PROGRAM_DATA );
FILE_DEFINITIONS : FILE
  = 1<LOCAL_FILE_DEFINITIONS + SEGMENT_DEFINITIONS>MAXIMUM_JOBS;
GLOBAL_FILE_ATTRIBUTES : FILE
  = 1< FILE_PARAMETERS >MAX_FILES_PER_JOB;
GLOBAL_LABEL_ATTRIBUTES : FILE
  = 1< LABEL_PARAMETERS >MAX_FILES_PER_JOB;
JOB_DATA : FILE
  = INPUT_FILE + OUTPUT_FILE;
JOB_DEFINITIONS : FLOW
  = JOB_LIMITS + INITIAL_TASK_ENVIRONMENT + GLOBAL_PROGRAM_DESCRIPTION
    + TASK_SERVICES_ENTRY_POINTS + EXECUTING_TASK
    + PRIMARY_TASK_LIST + DISPATCHABLE_TASK_LIST + EXCHANGE_PACKAGES
    + GENERAL_WAIT;
KERNEL_REQUEST : FLOW
  = ( LOCAL_FILE_REQUEST | PF_REQUEST | LOGGING_REQUEST
    | MESSAGE_COMMUNICATION_REQUEST | JOB_REQUEST
    | SEGMENT_REQUEST | PROGRAM_REQUEST );
KERNEL_RESPONSE : FLOW
  = ( LOCAL_FILE_RESPONSE | PF_RESPONSE | LOGGING_RESPONSE
    | MESSAGE_COMMUNICATION_RESPONSE | JOB_RESPONSE
    | SEGMENT_RESPONSE | PROGRAM_RESPONSE);
KNOWN_FILE_TABLE : FILE
  = 1<♦FILE_IDENTIFIER
    + ( FAT    ( TRANSIENT SEGMENT CASE )
      | FAT + CURRENT_FILE_STATUS  ( BAM DISK FILE CASE )
      | CURRENT_FILE_STATUS ( BAM TAPE/TERMINAL FILE CASE )
      )> MAX_FILES_PER_JOB;
LINK_DATA : FLOW
  = ( INTERACTIVE_DATA | 170_PF_DATA | OPERATOR_DATA | JOB_DATA
    | JOB_MESSAGE_DATA);
LOCAL_FILE_DEFINITIONS : FILE
  = KNOWN_FILE_TABLE + GLOBAL_LABEL_ATTRIBUTES + GLOBAL_FILE_ATTRIBUTES
    + FILE_MEDIUM_TABLE + STATIC_FILE_DESCRIPTION;
LOCAL_FILE_REQUEST : FLOW
  = ( BAM_REQUEST | FILE_MEDIUM_REQUEST | SWAP_WORKING_SET
    | FILE_REQUEST | LABEL_REQUEST | INITIALIZE_VOLUME_REQUEST
    | OPERATOR_ASSIGN_REQUEST | RESOURCE_REQUEST);
LOCAL_FILE_RESPONSE : FLOW
  =   BAM_RESPONSE;
MAXIMUM_JOBS : ELEMENT
  = 1 .. 4096;
MAXIMUM_TASKS_PER_JOB : ELEMENT
  = 20;
MAX_FILES_PER_JOB : ELEMENT
  = 1 .. 512;
PMAF_FILE_DATA : FLOW
  = TARGET_OBJECT_TEXT + COEXISTENT_MODULES + TRANSFER_SYMBOL_PARAMETER
    + RESTRUCTURED_PROGRAM + PROGRAM_PROFILE;
PROCESS_SEGMENT_TABLE : FILE
  = ♦SEGMENT_NUMBER + <SEGMENT_DESCRIPTOR + FILE_ID>4096;
PROCESS_SEGMENT_TABLES : FILE
  = ♦LOCAL_TASK_ID + 1<PROCESS_SEGMENT_TABLE>MAXIMUM_TASKS_PER_JOB;
PROGRAM_DATA : FILE
  = OBJECT_INFORMATION + LOAD_MAP;
```

```
SB_DATA : FLOW
{ DATA STORED IN SYSTEM BUFFERS INACCESSABLE TO USER PROGRAM }
  = ;
SEGMENT_DEFINITIONS : FILE
  = <PROCESS_SEGMENT_TABLES>;
STACKED_CONDITION_DATA : FLOW
  = STACK_FRAMES + [ESTABLISHED_DESCRIPTORS];
STATIC_FILE_DESCRIPTION : FILE
  = 1< SYSTEM_FILE_LABEL > MAX_FILES_PER_JOB;
USER_ADDRESS_SPACE : FILE
  = LOADED_PROGRAM + STACKED_CONDITION_DATA + BUFFERED_DATA
    + USER_DATA + INSTRUCTIONS;
USER_DATA : FLOW
{ DATA ACCESSABLE TO USER PROGRAM }
  = ;
USER_EXIT_REQUEST : FLOW
  = ( CONDITION_HANDLER_REQUEST | BAM_EXIT_REQUEST );
USER_EXIT_RESPONSE : FLOW
  = CONDITION_HANDLER_RESPONSE;
USER_KERNEL_REQUEST : FLOW
{ THESE ARE THE REQUESTS THAT THE COMMAND LANGUAGE INTERPRETER ISSUES
{   AS A RESULT OF INTERPRETING USER COMMANDS.}
  = KERNEL_REQUEST;
USER_KERNEL_RESPONSE : FLOW
  = KERNEL_RESPONSE;
```

3.1 MANAGE_LOCAL_FILES

Note:

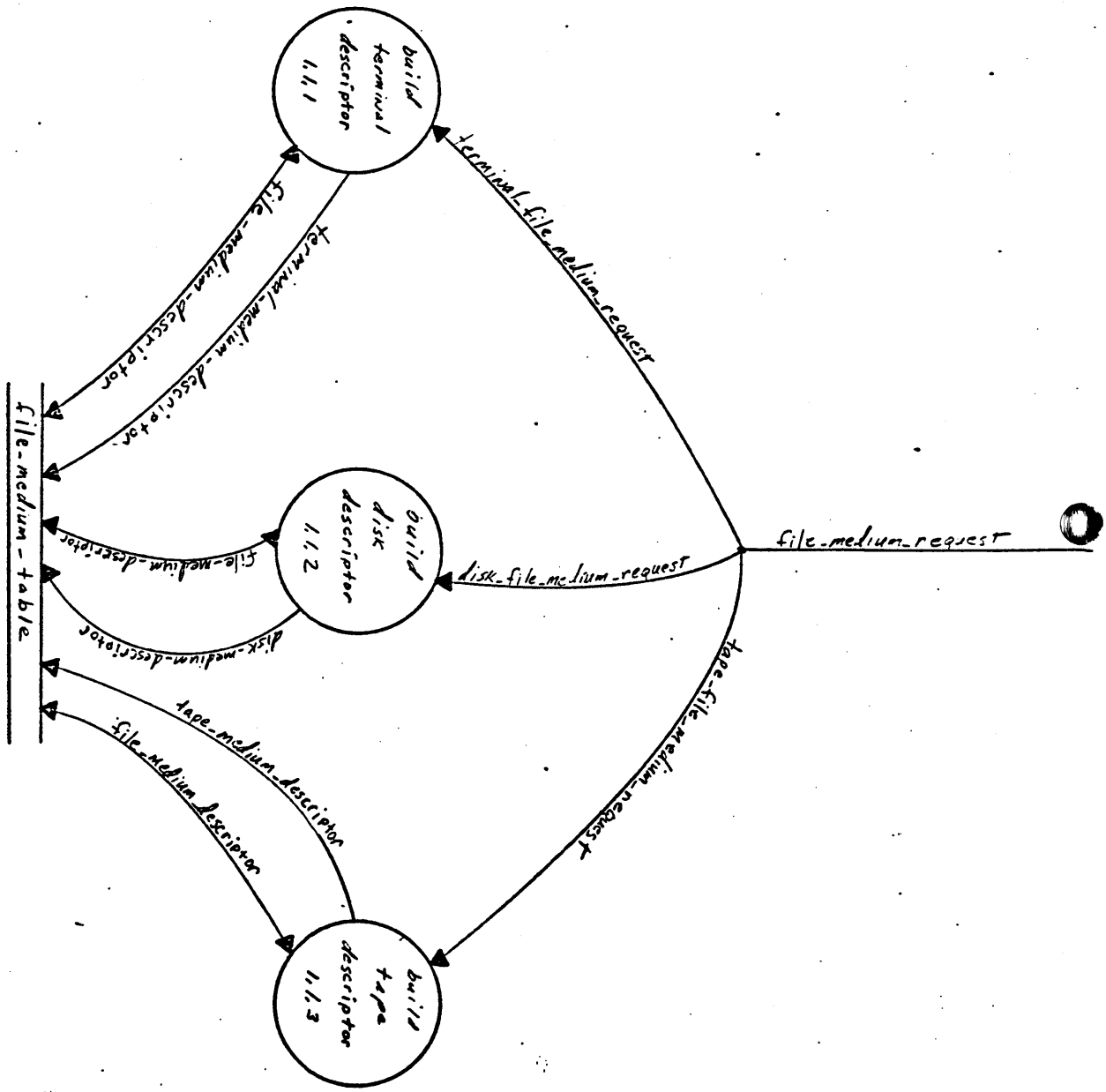
Diagram No: 1.0 Rev: 5 Diagram Name: MANAGE LOCAL FILES Superior Diagram Name: LEVEL 0 NOSIVE

DC DATA FLOW DIAGRAM WORKSHEET

Author: [illegible] Date: 3/2/79 Page of

Note:

Diagram No: 1.1 Rev: 2 Diagram Name: create w/ medium desc. Superior Diagram Name: manage mass files



DC DATA FLOW DIAGRAM WORKSHEET

Author: [illegible] Date: 2/27/79 Page of

Note:

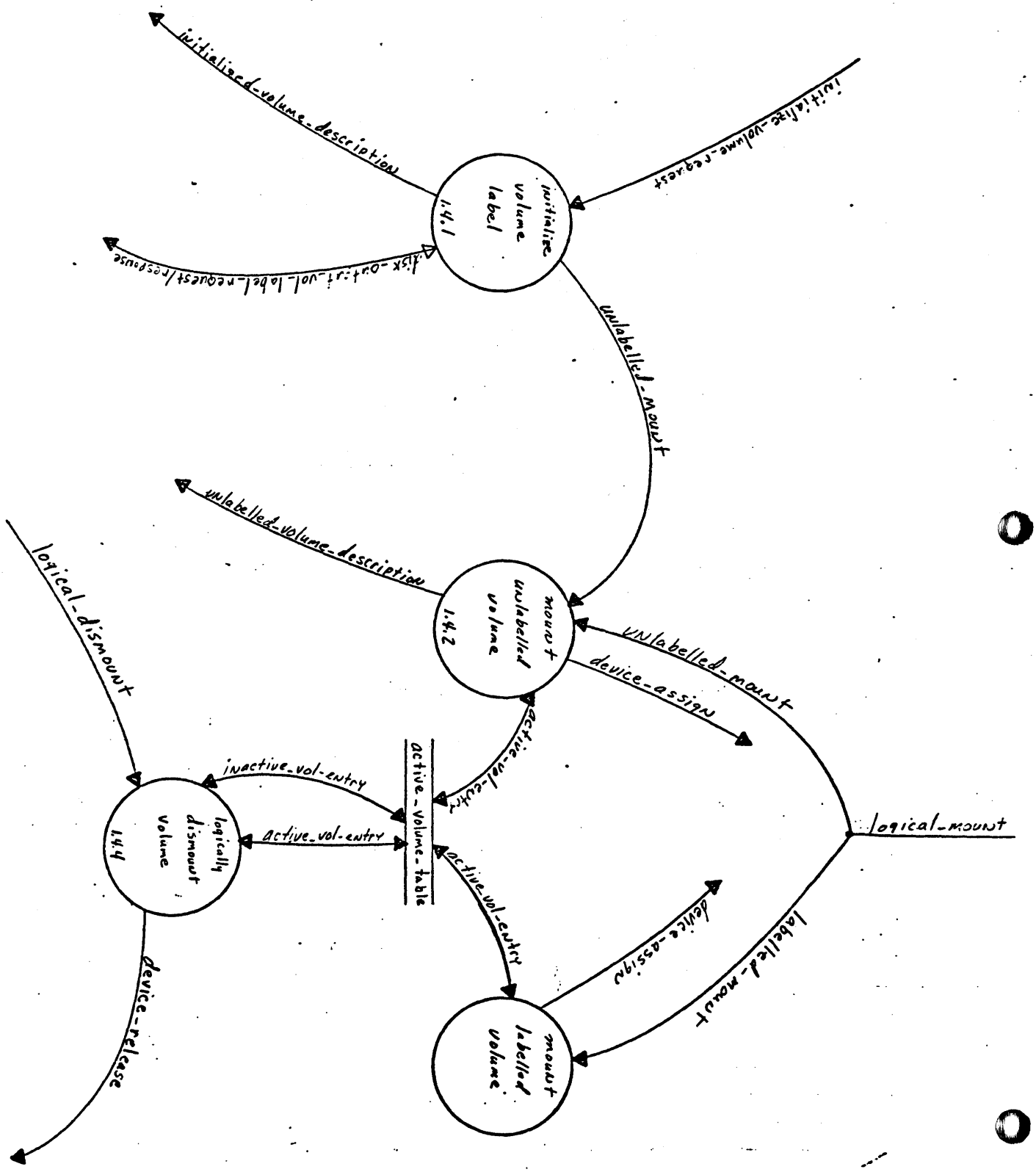
Diagram No: 1.3 Rev: 2 Diagram Name: MANAGE FILE ACCESS Superior Diagram Name: MANAGE LOCAL FILES

DC DATA FLOW DIAGRAM WORKSHEET

Author: AL LAWSON Date: 3/2/79 Page of

Note:

Diagram No: 1.4 Rev: 3 Diagram Name: MAN. ISS.; ONLINE DEVICES Superior Diagram Name: MANAGE LOCAL FILES



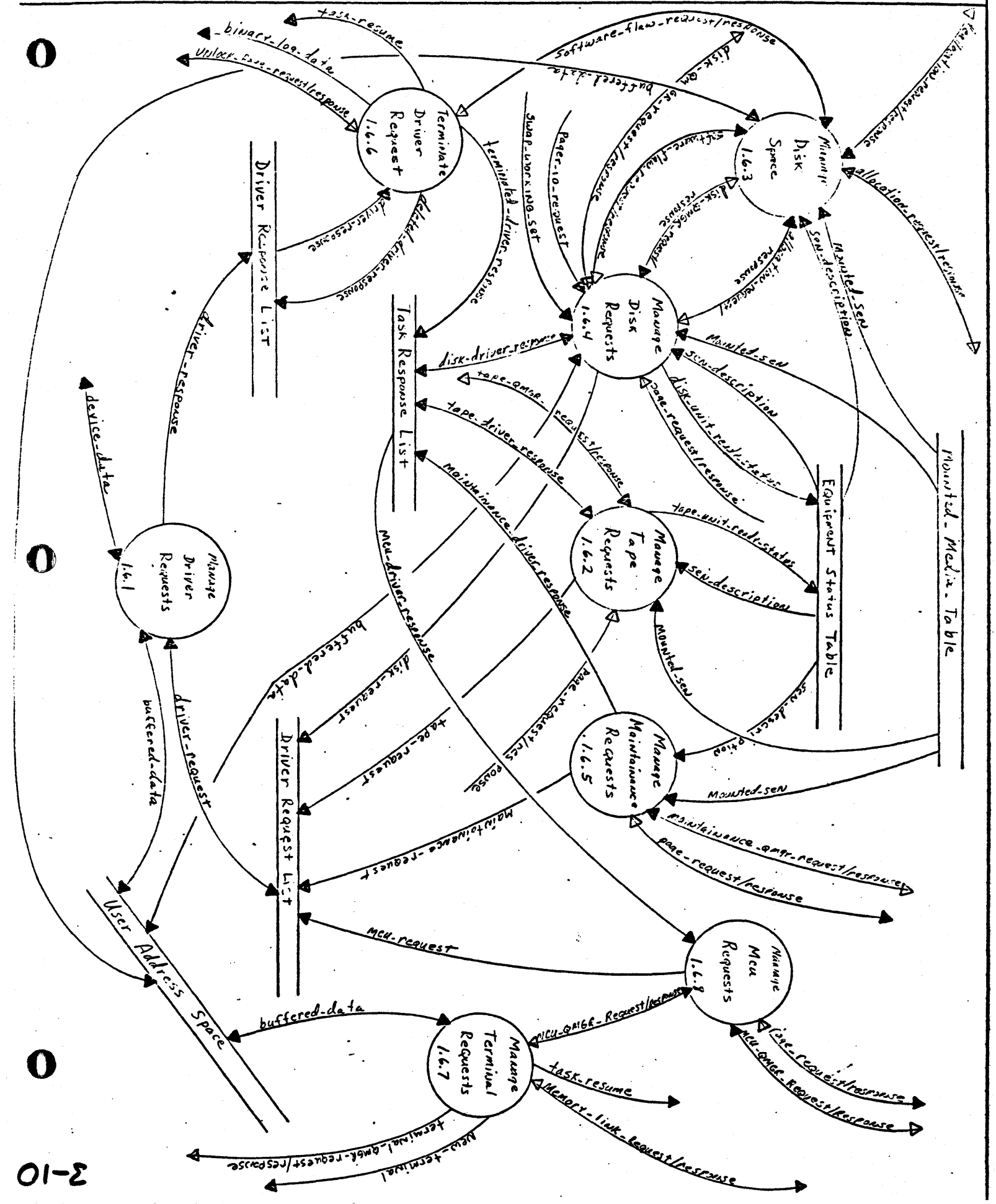
1 of 5

DC DATA FLOW DIAGRAM WORKSHEET

Author: [illegible] Date: [illegible] Page of

Note:

Diagram No: 1.6 Rev: 4 Diagram Name: Maintain Peripheral Devices Superior Diagram Name: Manage User Jobs



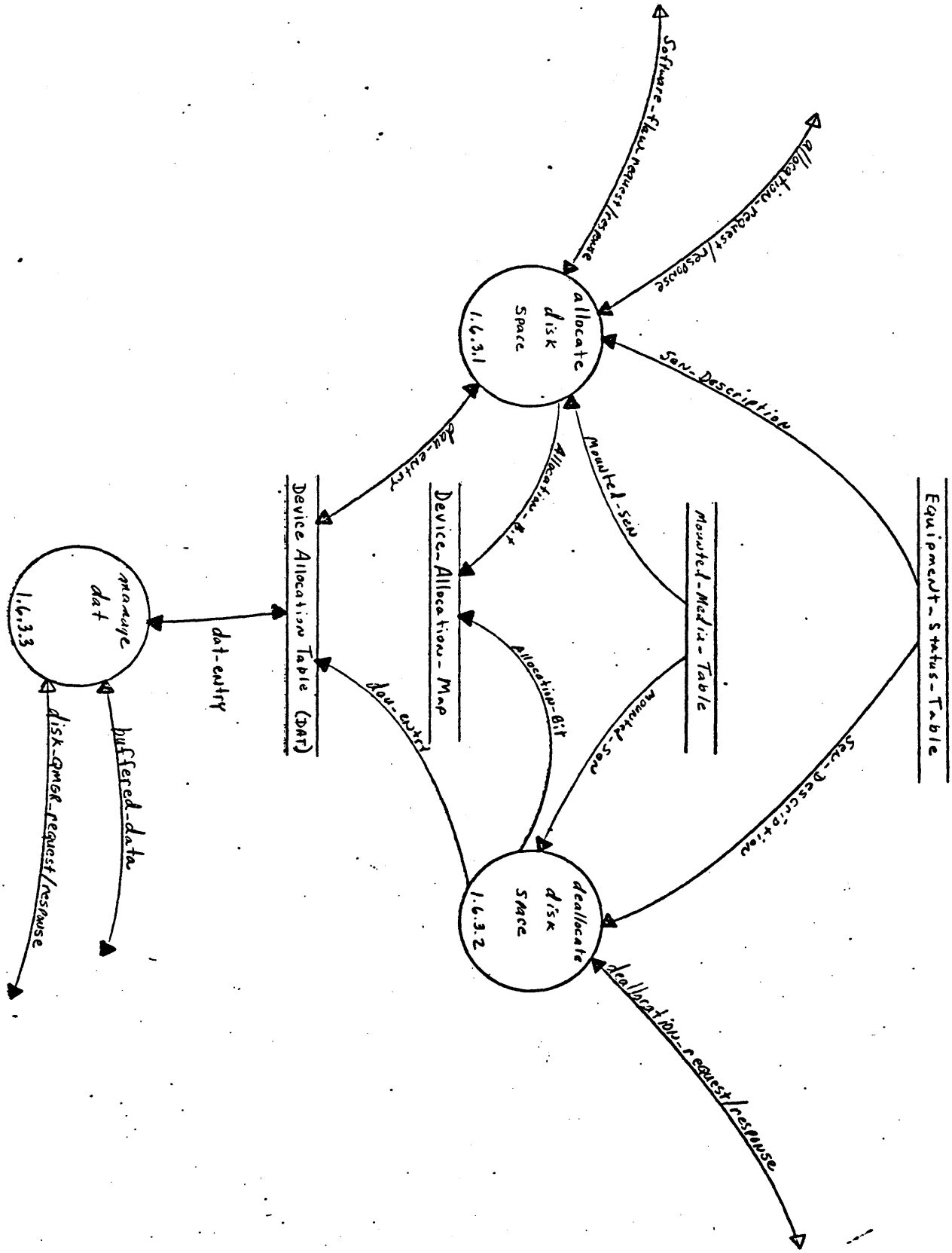
3-10

C DATA FLOW DIAGRAM WORKSHEET

hor: A. LARSON Date: 3/2/79 Page of

e:

gram No: 1.6.3 Rev: 4 Diagram Name: MANAGE DISK SPACE Superior Diagram Name: MAINTAIN PHYSICAL RESOURCES

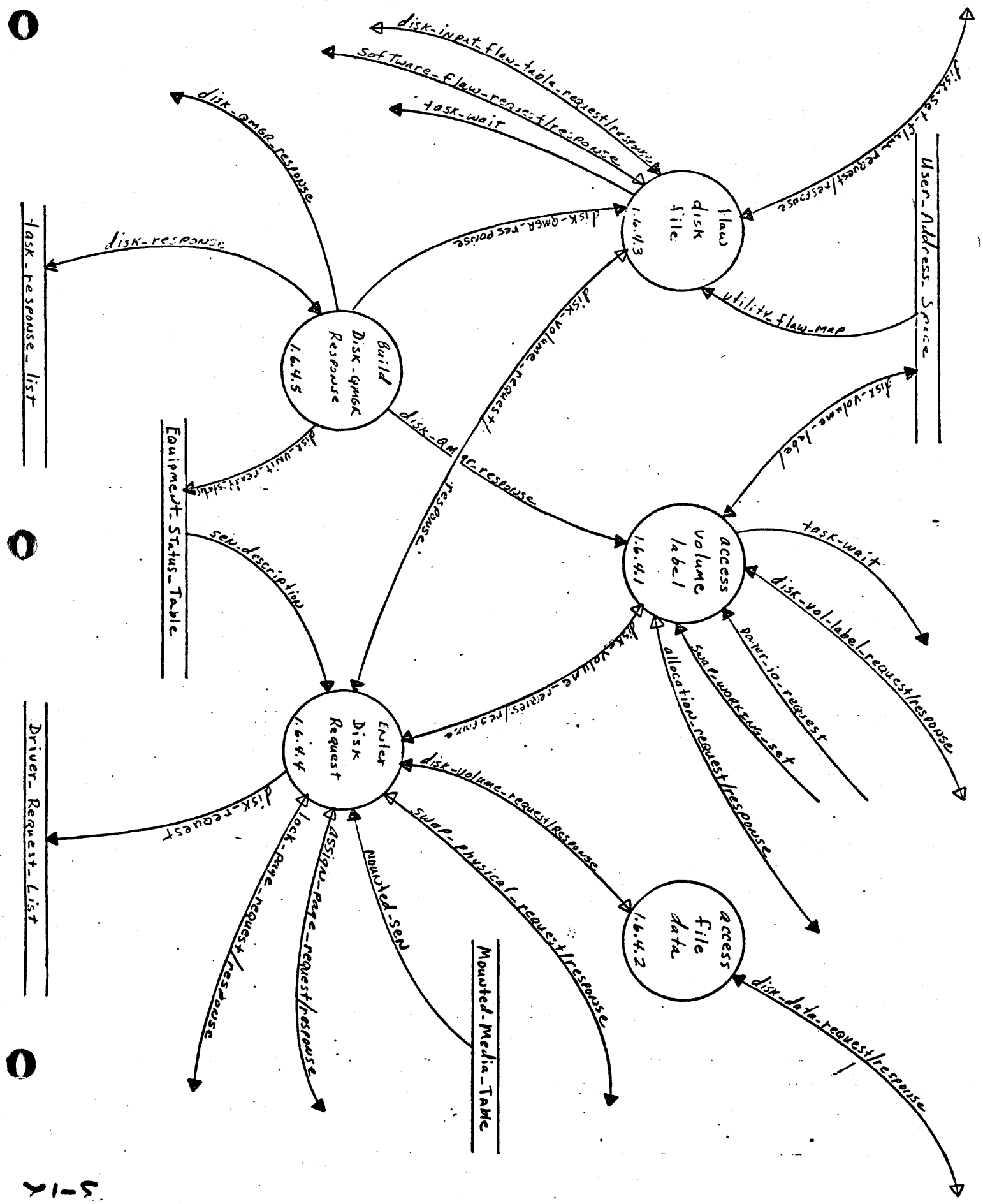


DC DATA FLOW DIAGRAM WORKSHEET

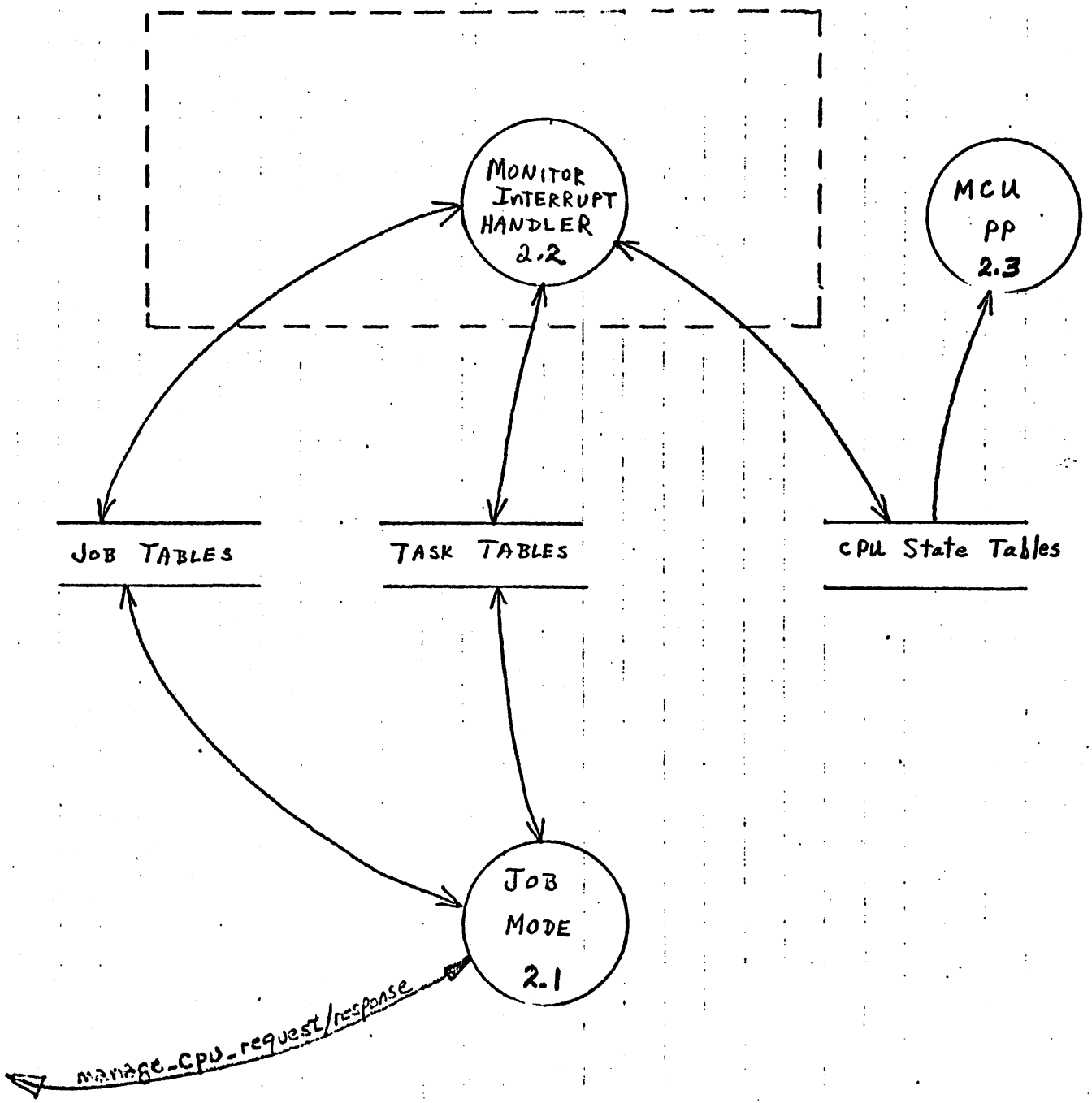
Author: [illegible] Date: 3/[illegible]/79 Page of

Note:

Diagram No: 1.6.4 Rev: 3 Diagram Name: Manage Disk Resource Superior Diagram Name: Manage Physical Devices



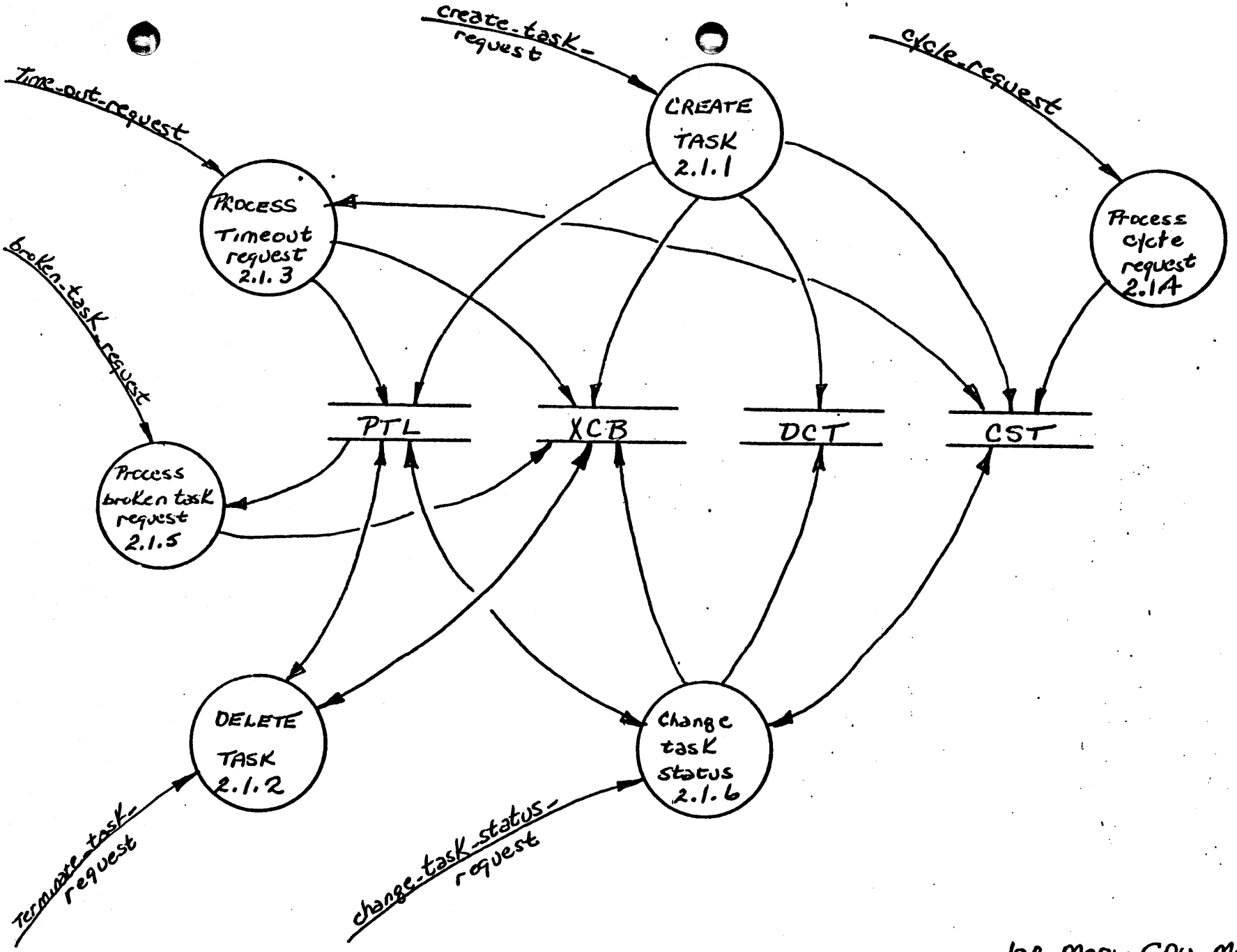
3.2 MANAGE_CPU



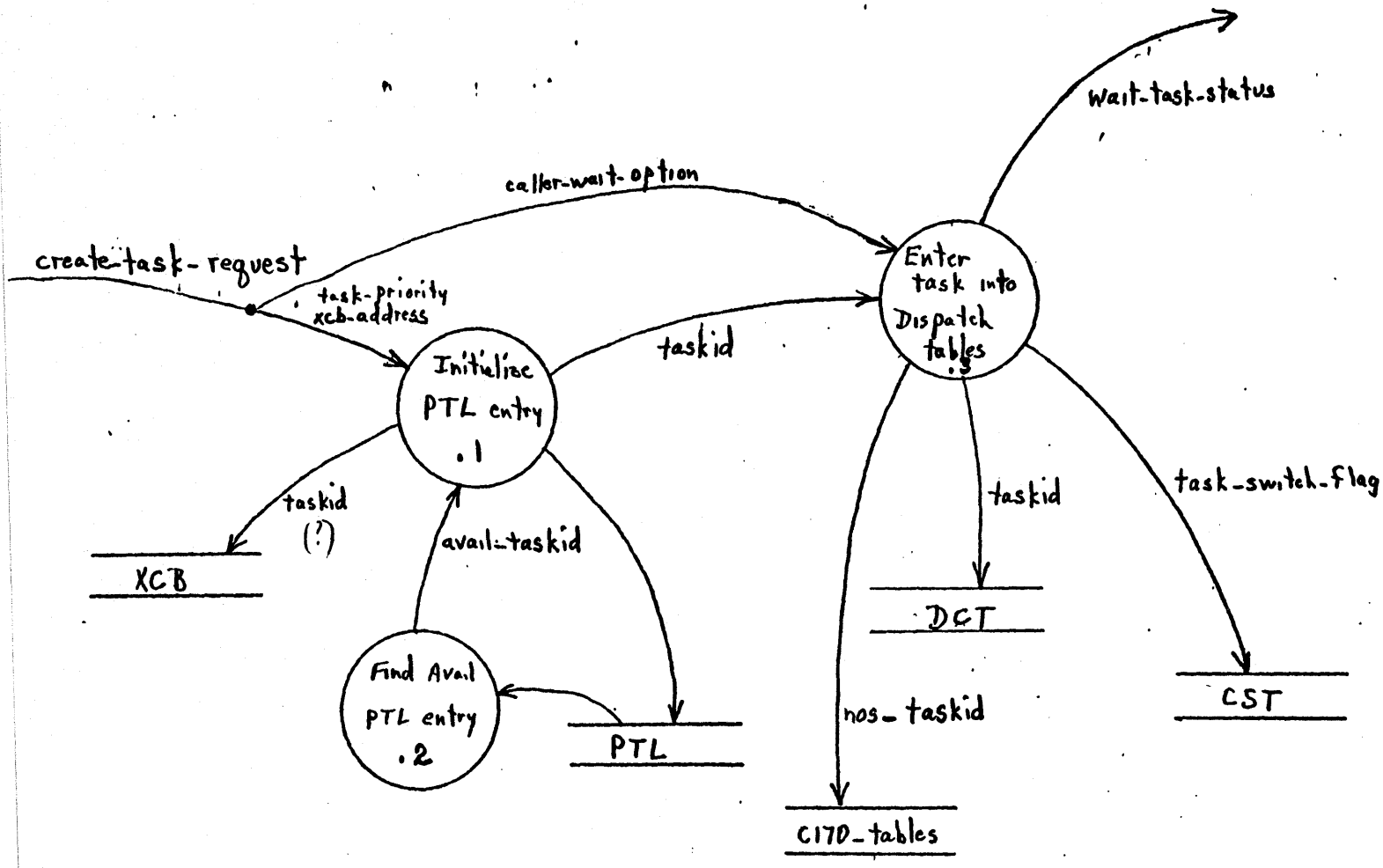
Level 1: MANAGE CPU.
2.0

2/28/79

W-5C



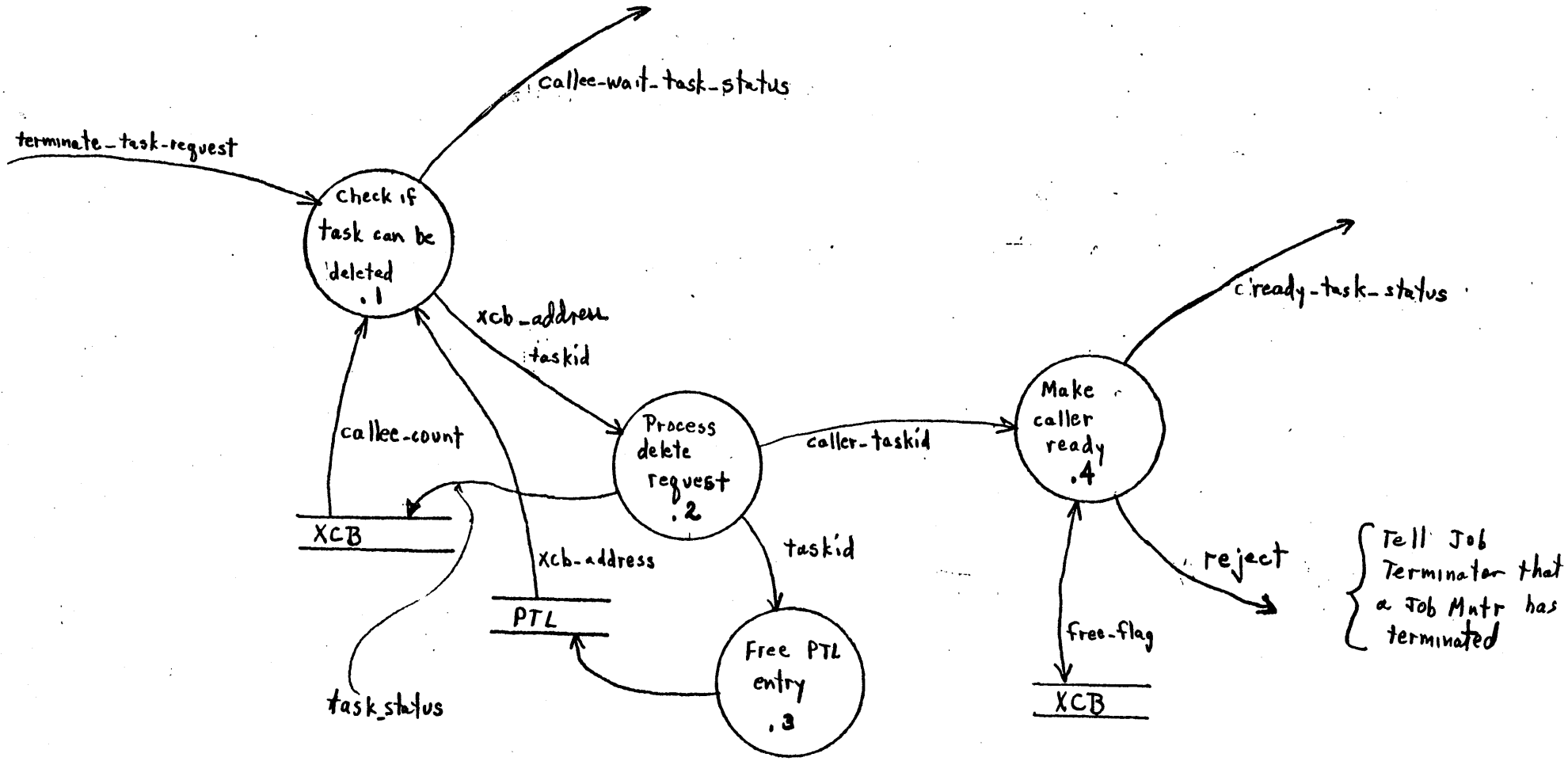
JOB MODE CPU MANAGEMENT
2.1
2/28/79



?

2.1.1 CREATE_TASK request

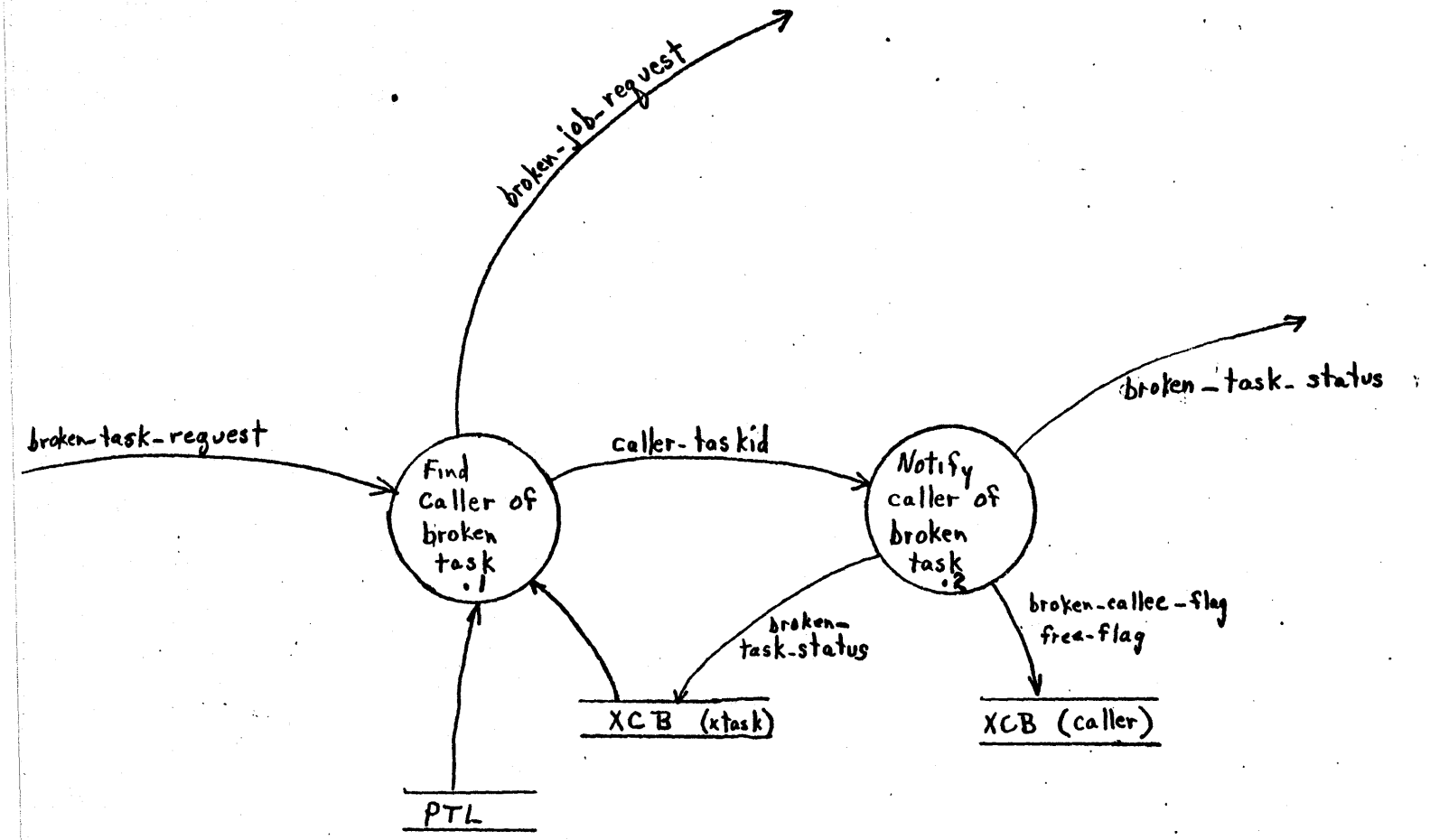
2/28/79



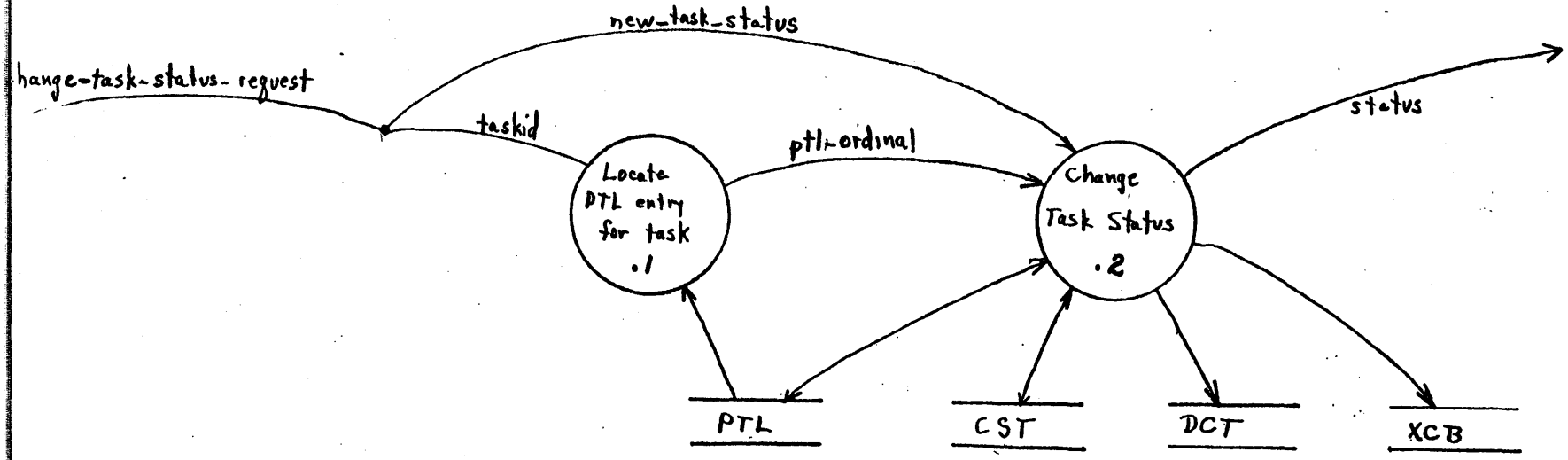
2.1.2 DELETE TASK REQUEST

2/28/79

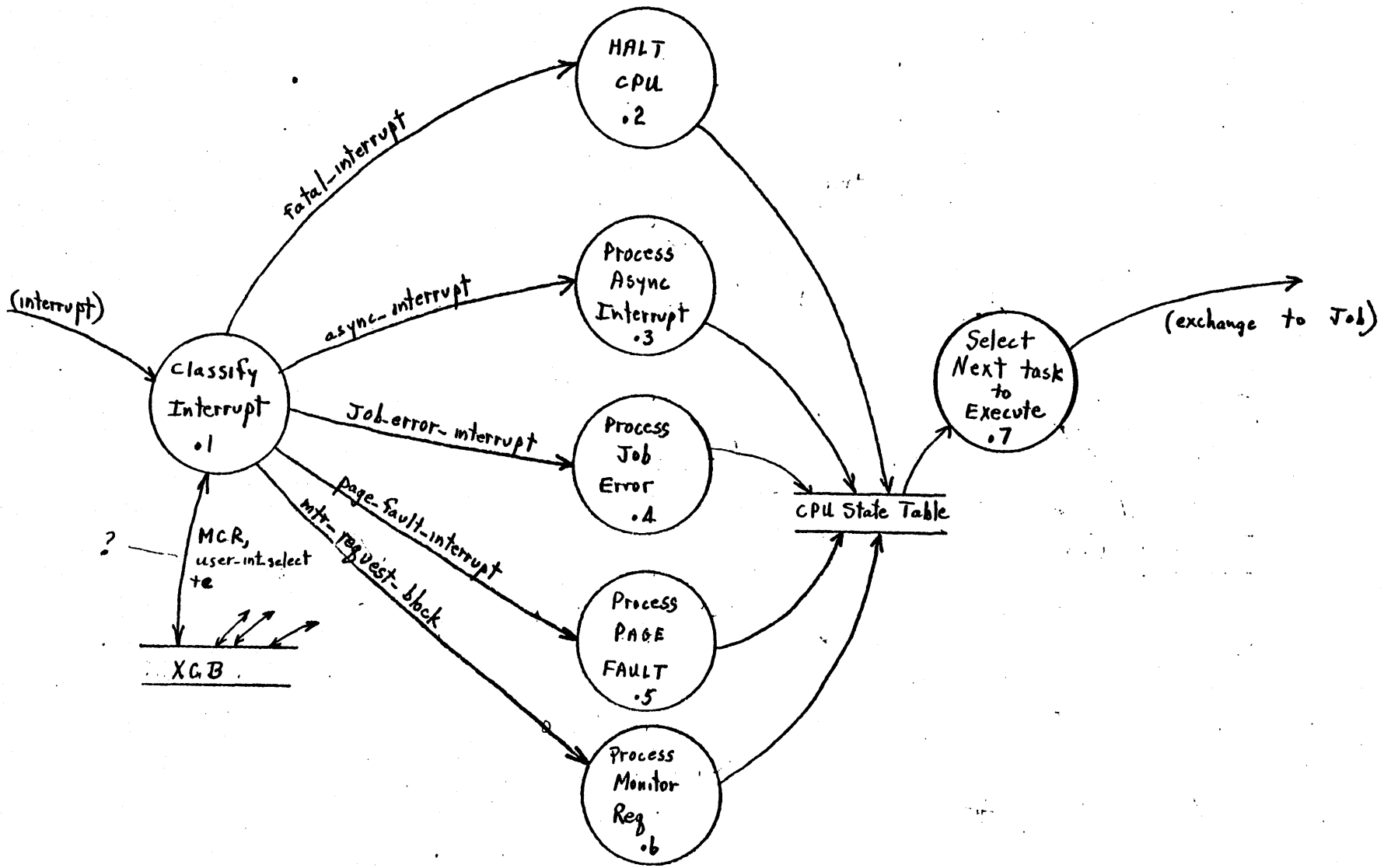
WWW



2.1.5 BROKEN TASK REQUEST
2/28/79

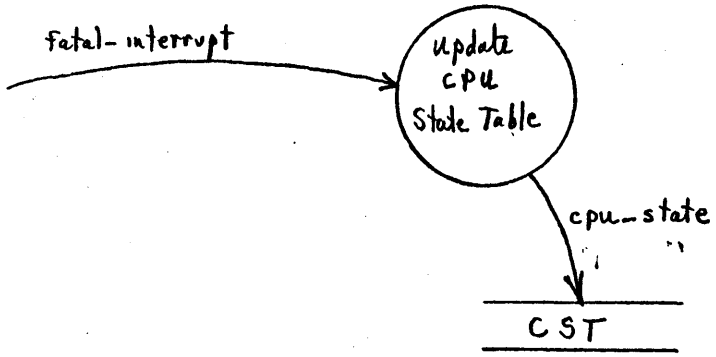


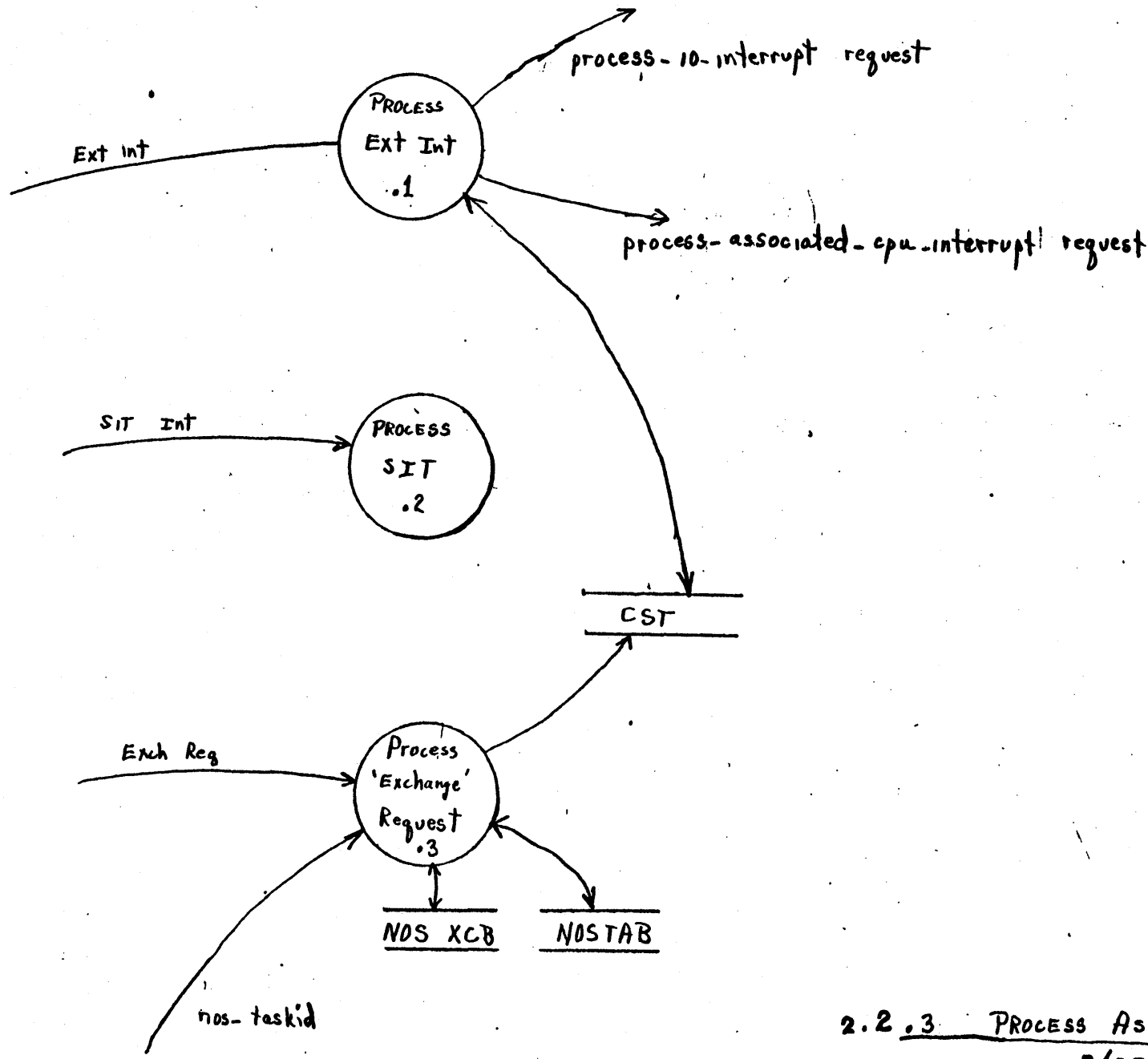
2.1.6 CHANGE_TASK_STATUS Request
2/28/79



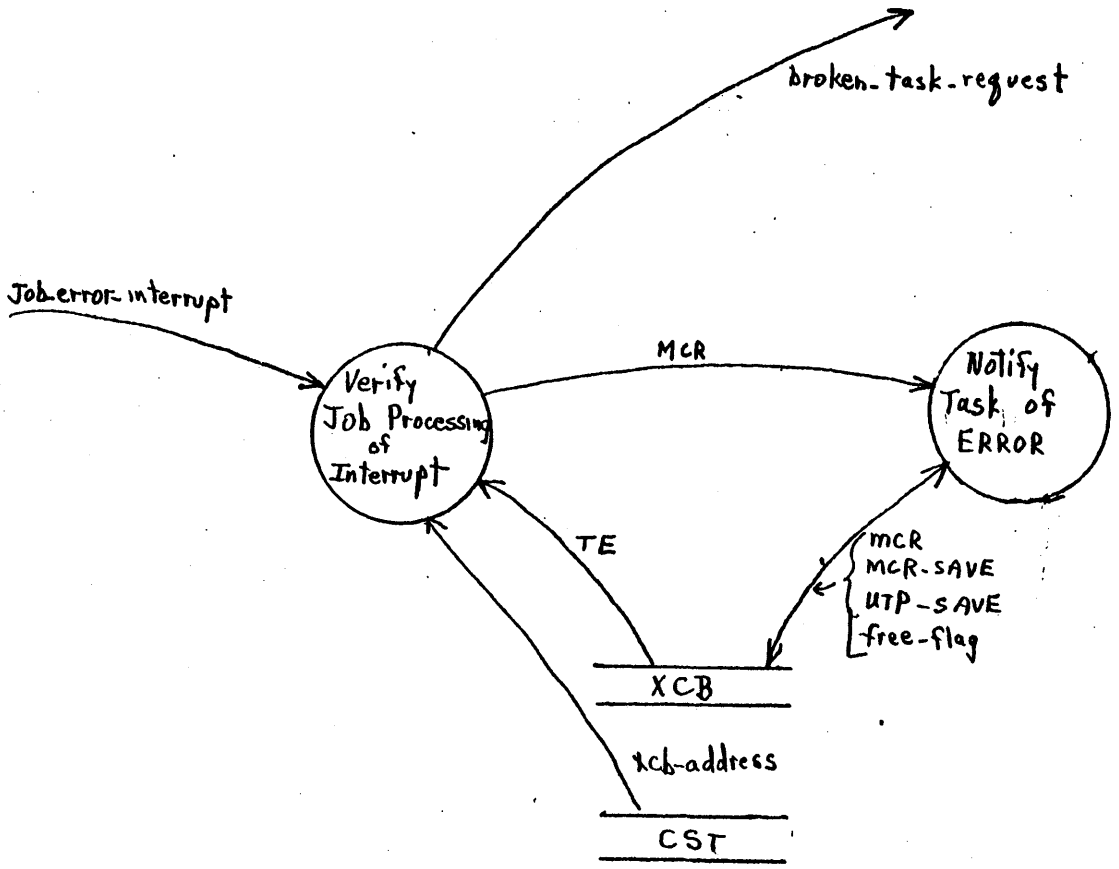
2.2 MONITOR INTERRUPT HANDLER

2/28/79

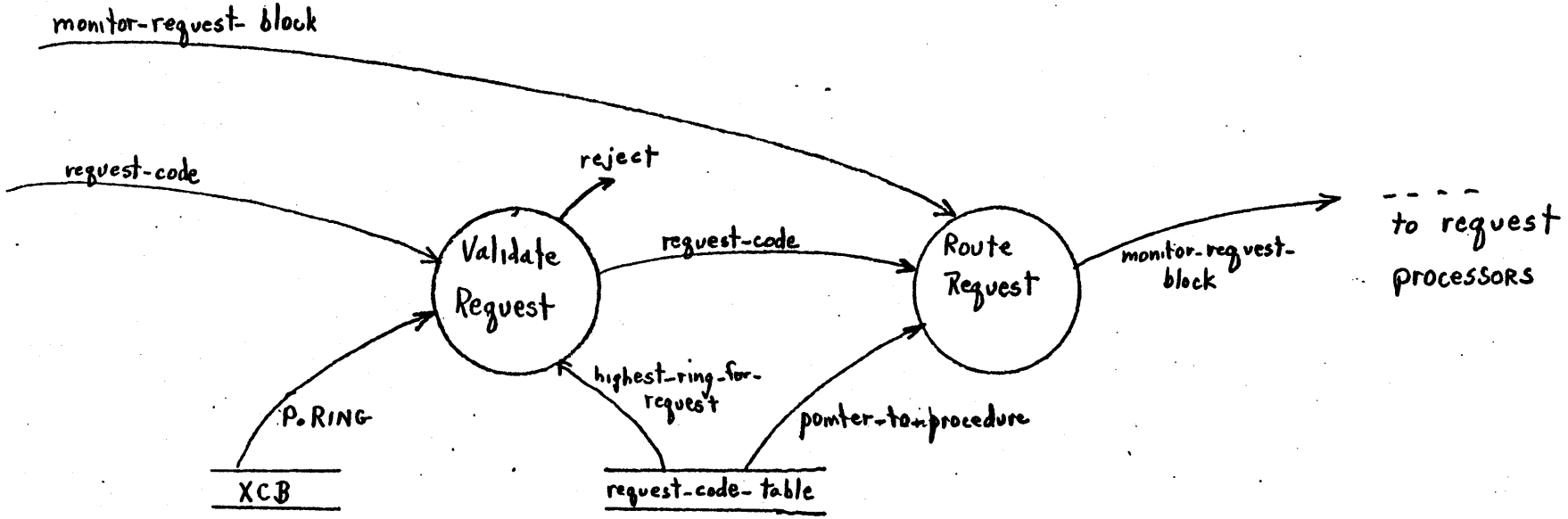

fatal_interrupt
update
cpu
State Table
cpu_state
CST



2.2.3 PROCESS ASYNC INTERRUPT
2/28/79



2.2.4 Process Job Error
2/28/79


monitor-request- block
request-code
reject
Validate Request
request-code
Route Request
monitor-request-block
to request processors
highest-ring-for-request
P. RING
pointer-to-procedure
XCB
request-code- table

OUT 79/02/26. 08.31.16.

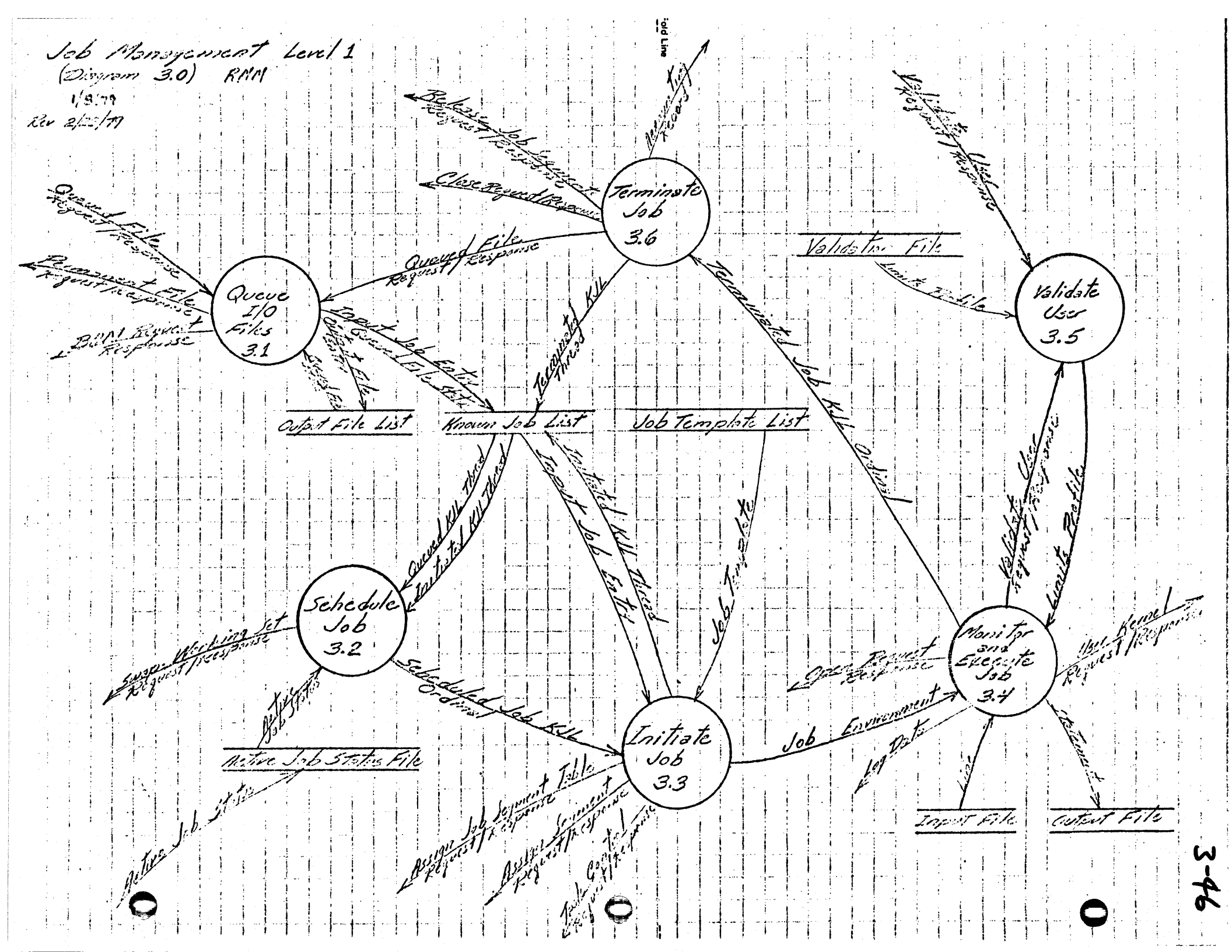
```
async_interrupt : FLOW
  = [sit_interrupt]
    + [exchange_interrupt]
    + [external_interrupt];
avail_taskid : FLOW
  = taskid;
broken_job_request : FLOW
  = running_job_ordinal;
broken_task_request : FLOW
  = taskid;
caller_taskid : FLOW
  = taskid;
caller_wait_option : FLOW
  = (wait
    ! nowait);
change_task_status_request : FLOW
  = taskid
    + task_status;
cpu_request : FLOW
  = (create_task_request
    ! delete_task_request
    ! timeout_request
    ! cycle_request);
cpu_state : FLOW
  = (cpu_running
    ! cpu_stopped);
cpu_state_table : FILE
  = <cpu_state
```

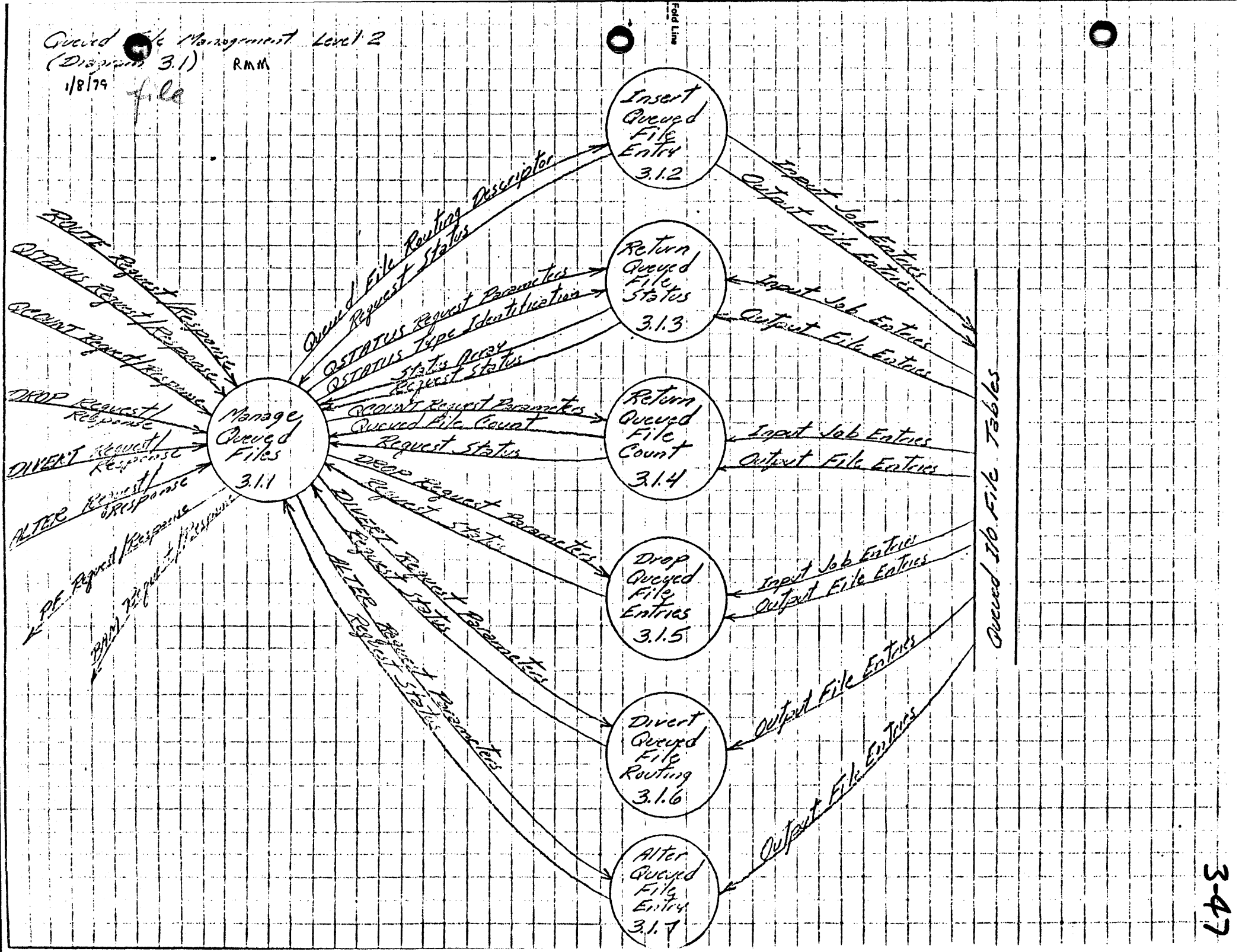
```
    + task_switch_flag
    + xtask_xcb_address
    + xtaskid>;
create_task_request : FLOW
  = xcb_address
    + caller_wait_option
    + task_priority;
cycle_request : FLOW {give up control to another task}
  = ;
delete_task_request : FLOW
  = task_id;
end_timeout : FLOW
  = free_running_clock_value;
fatal_interrupt : FLOW
  = [processor_malfunction]
    + [power_warning]
    + [memory_malfunction];
job_error_interrupt : FLOW
  = [instruction_spec_error]
    + [address_spec_error]
    + [access_violation]
    + [env_spec_error]
    + [invalid_segment]
    + [out_call_in_return]
    + [priv_instruction_fault]
    + [unimplemented_instruction]
    + [inter_ring_pop]
    + [critical_frame_flag];
```

```
monitor_condition_register : FLOW
  [fatal_interrupt]
    + [job_error_interrupt]
    + [async_interrupt]
    + [page_fault_interrupt]
    + [system_call_interrupt];
primary_task_list : FILE
  = <primary_task_list_entry>;
primary_task_list_entry : FILE
  = *taskid
    + xcb_address
    + running_job_ordinal
    + task_status
    + task_queue_link
    + task_queue_head_address
    + task_priority;
request_code_table : FILE
  = request_code
    + high_ring_for_request
    + pointer_to_procedure;
taskid : FLOW
  = ptl_ordinal
    + sequence_number;
task_queue_head_address : FLOW
  = pva;
task_queue_link : FLOW
  ptl_ordinal;
task_status : FLOW
```

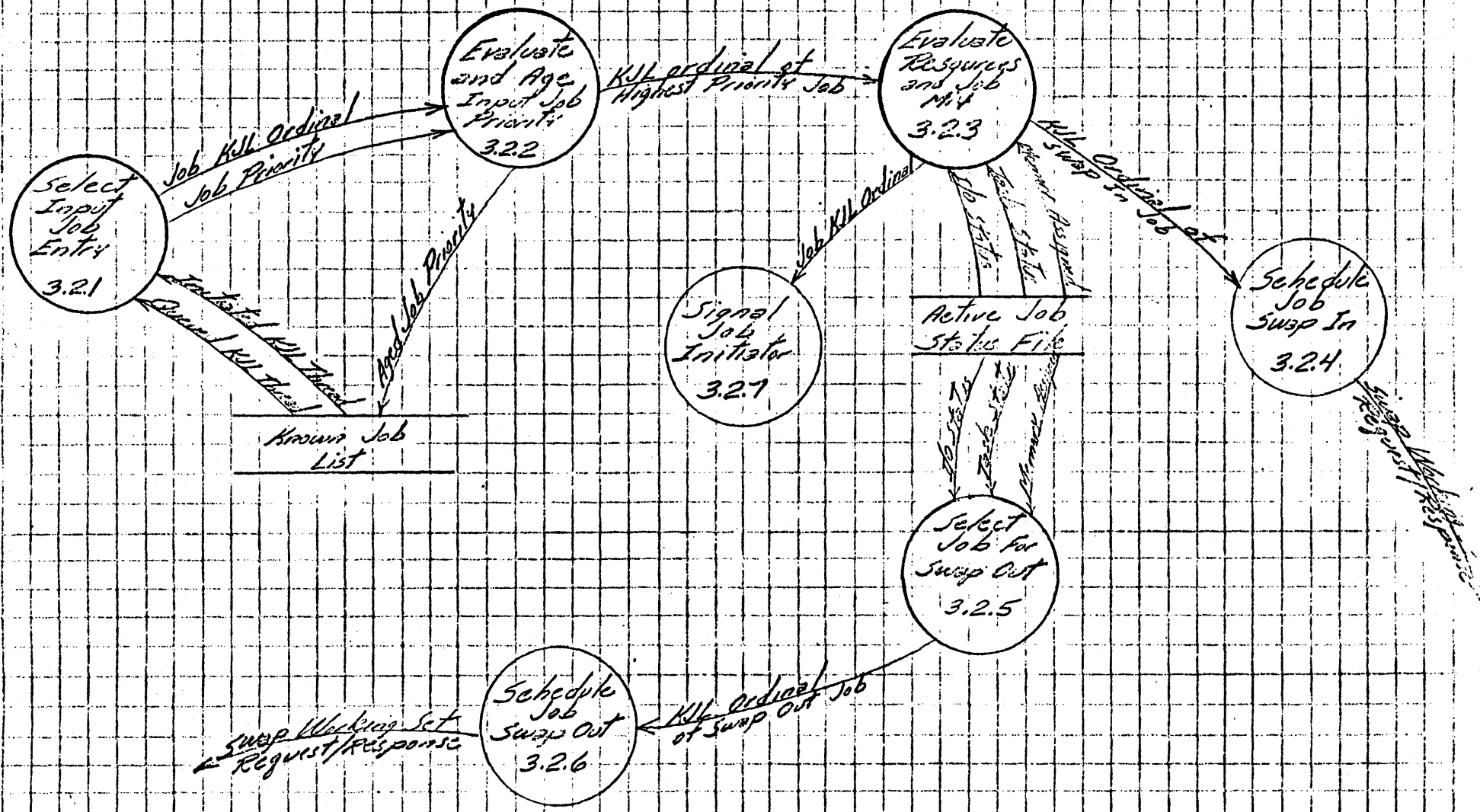
```
  = (ready_task_status
    ! wait_task_status
    ! timeout_task_status
    ! memory_wait_task_status
    ! page_wait_task_status
    ! callee_wait_task_status
    ! broken_task_status
    ! terminated_task_status);
timeout_request : FLOW
  = free_running_clock_value;
xcb : FILE (execution_control_block)
  = exchange_package
    + user_interrupt_selections
    + task_accounting_information
    + task_status
    + taskid
    + caller_taskid
    + callee_count
    + broken_callee_flag
    + callee_terminated_flag
    + end_timeout;
:taskid : FLOW
  = taskid;
```

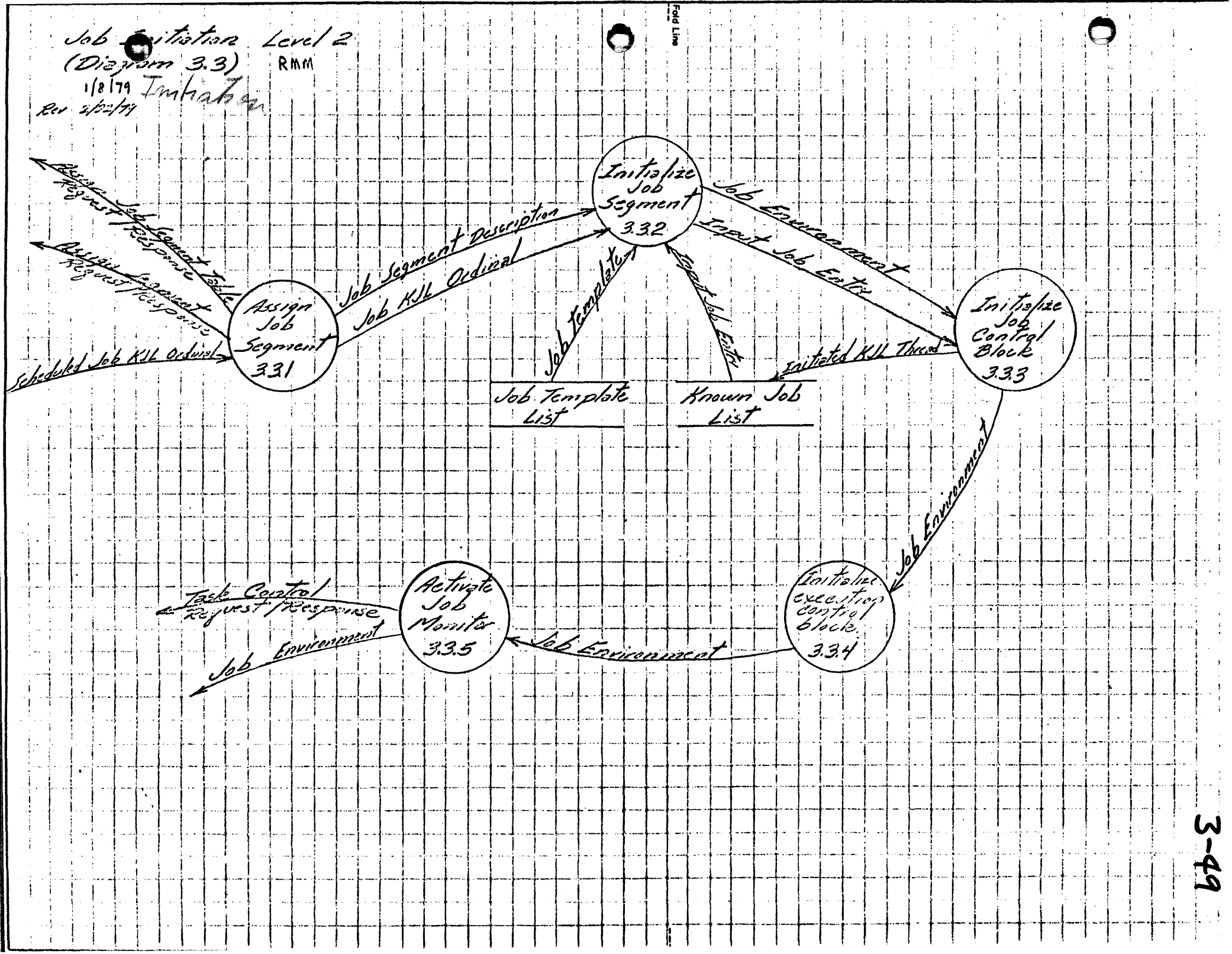
3.3 MANAGE JOBS


Job Management Level 1
(Diagram 3.0) RMM
1/9.79
Rev 2/22/79
Terminate Job 3.6
Queue I/O Files 3.1
Validate User 3.5
Schedule Job 3.2
Monitor and Execute Job 3.4
Initiate Job 3.3
Validation File
Output File List
Known Job List
Job Template List
Active Job Status File
Input File
Output File
Terminated Job Kill Original
Scheduled Job Kill Original
Job Environment
Job Data

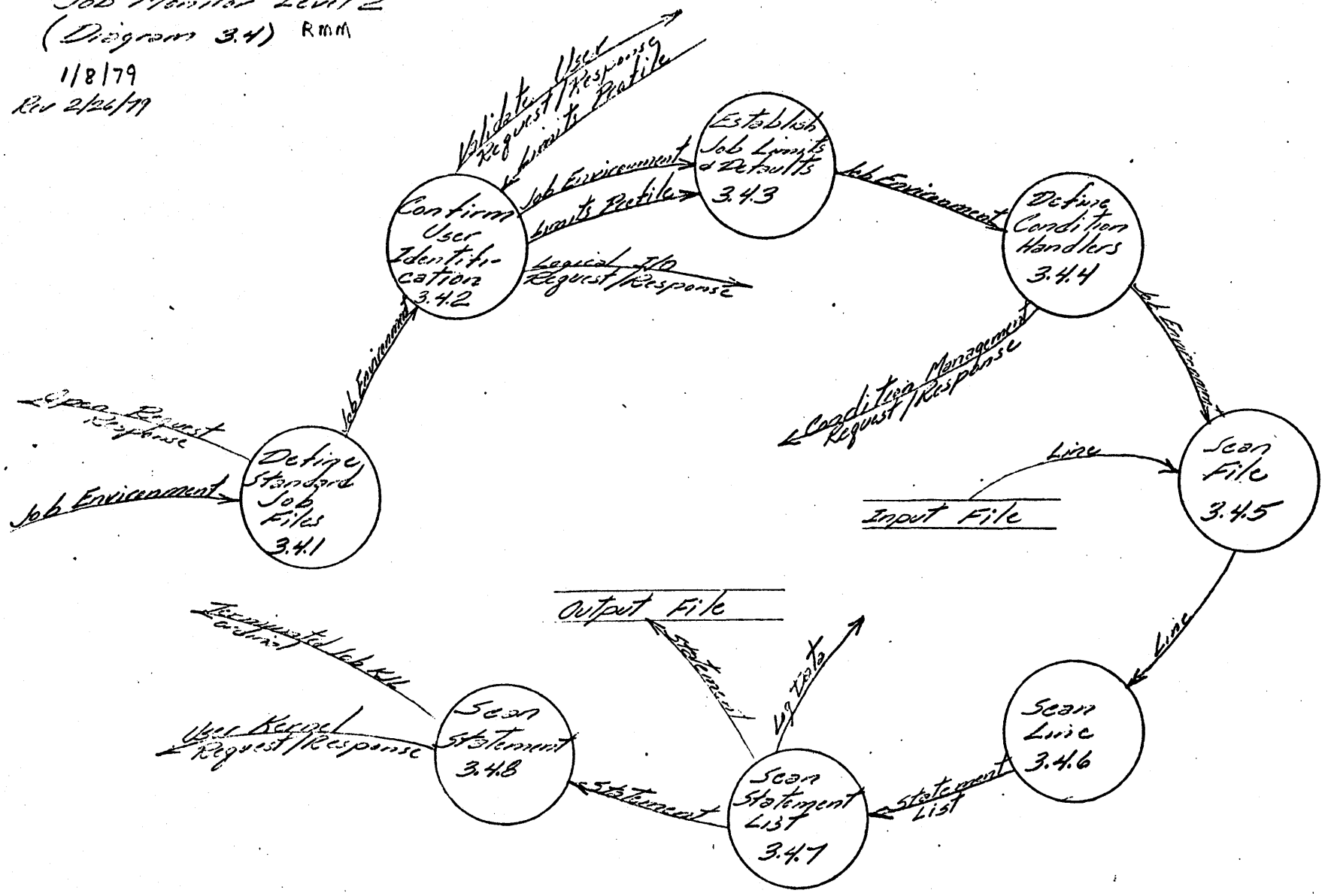

Queued File Management Level 2
(Diagram 3.1)
RMM
1/8/79
file
Fold Line
Insert Queued File Entry 3.1.2
Return Queued File Status 3.1.3
Return Queued File Count 3.1.4
Drop Queued File Entries 3.1.5
Divert Queued File Routing 3.1.6
Alter Queued File Entry 3.1.7
Manage Queued Files 3.1.1
Queued File Tables
Input Job Entries
Output File Entries
Queued File Routing Descriptor
Request Status
QSTATUS Request Parameters
QSTATUS Type Identification
Status Array
Request Status
QCOUNT Request Parameters
Queued File Count
Request Status
DROP Request Parameters
Request Status
DIVERT Request Parameters
Request Status
ALTER Request Parameters
Request Status
ROUTE Request/Response
QSTATUS Request/Response
QCOUNT Request/Response
DROP Request/Response
DIVERT Request/Response
ALTER Request/Response
PF Request/Response
RMM Request/Response

Job Scheduling Level 2
(Diagram 3.2) RMM
1/8/79
Rev. 2/22/79




Job Initiation Level 2
(Diagram 3.3)
1/8/79 Initiation
Rev 2/22/79
RMM
Fold Line
Assign Job Segment Request
Job Segment Table
Assign Job Segment Request Response
Scheduled Job KJL Ordinal
Assign Job Segment 331
Job Segment Description
Job KJL Ordinal
Initialize Job Segment 3.3.2
Job Template
Known Job Entry
Job Environment
Input Job Entry
Initialize Job Control Block 3.3.3
Initiated KJL Thread
Job Template List
Known Job List
Job Environment
Initialize execution control block 3.3.4
Job Environment
Activate Job Monitor 3.3.5
Task Control Request Response
Job Environment

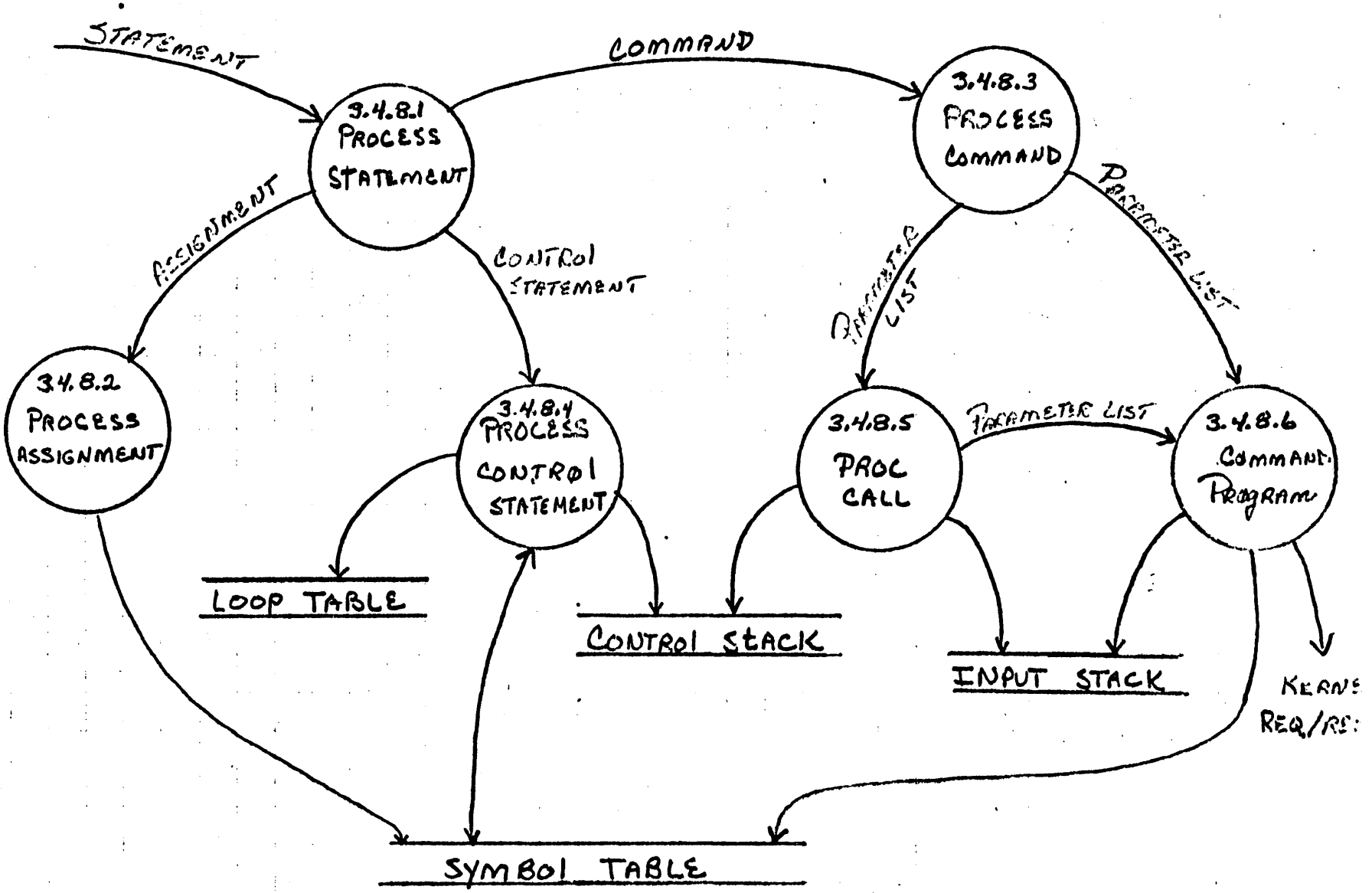
Job Monitor Level 2
(Diagram 3.4) RMM
1/8/79
Rev 2/26/79

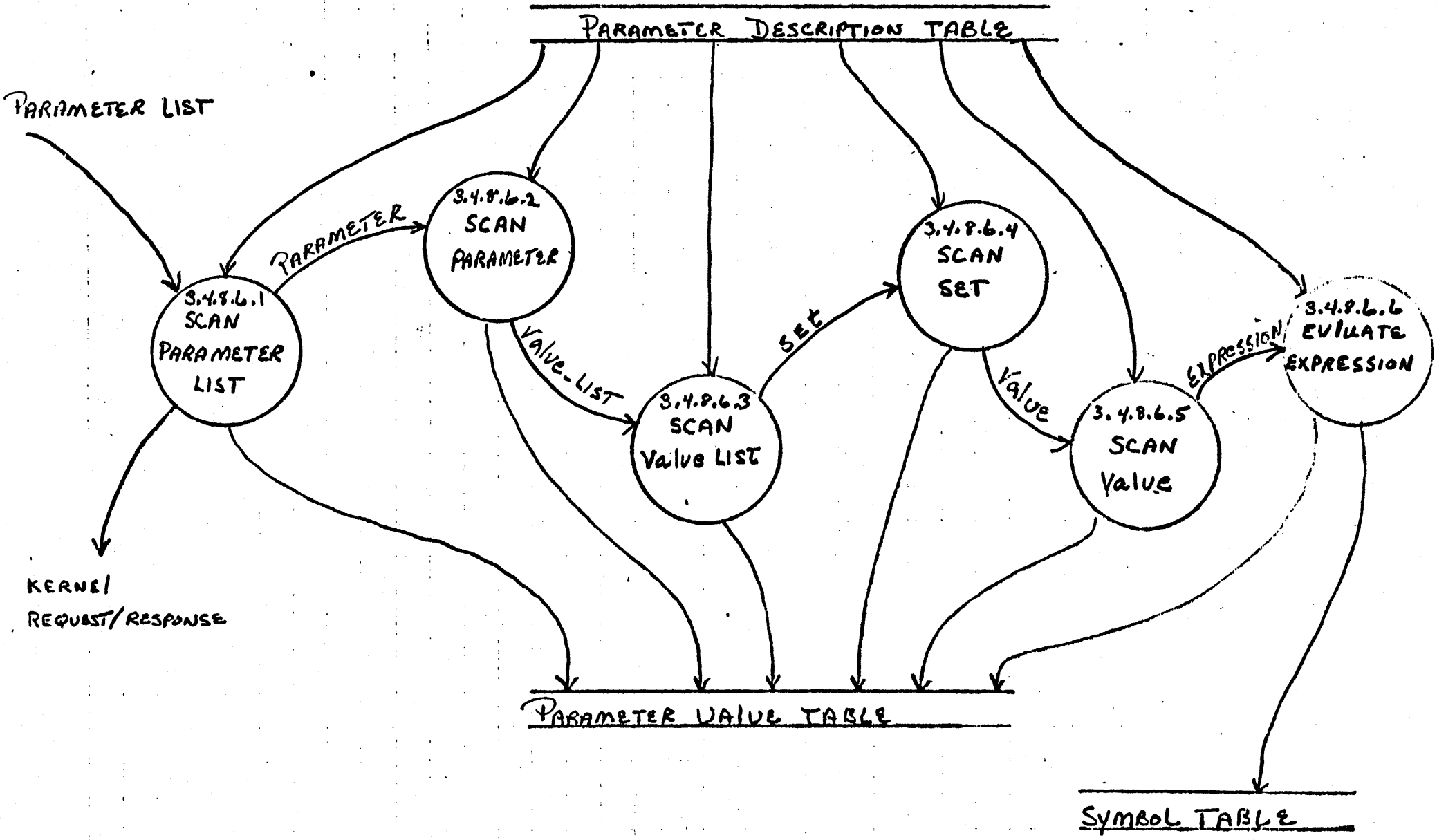


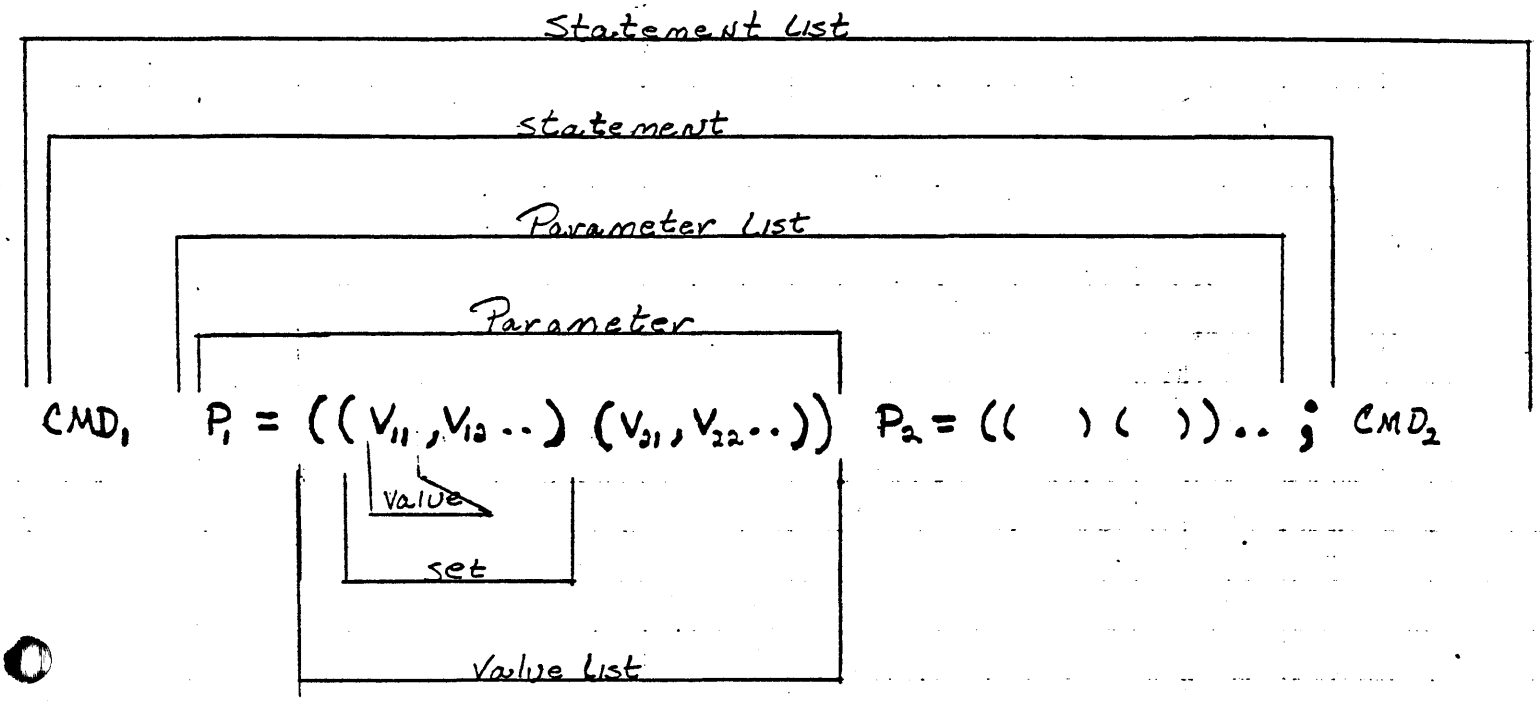
Monitor & Execute Job
Level 2
(Diagram 3.4)

SCL
A/77
CGN

SCP/I STATEMENT
Level 3




PARAMETER DESCRIPTION TABLE
PARAMETER LIST
3.4.8.6.1
SCAN
PARAMETER
LIST
PARAMETER
3.4.8.6.2
SCAN
PARAMETER
VALUE-LIST
3.4.8.6.3
SCAN
Value LIST
SET
3.4.8.6.4
SCAN
SET
VALUE
3.4.8.6.5
SCAN
Value
EXPRESSION
3.4.8.6.6
EVALUATE
EXPRESSION
KERNEL
REQUEST/RESPONSE
PARAMETER VALUE TABLE
SYMBOL TABLE



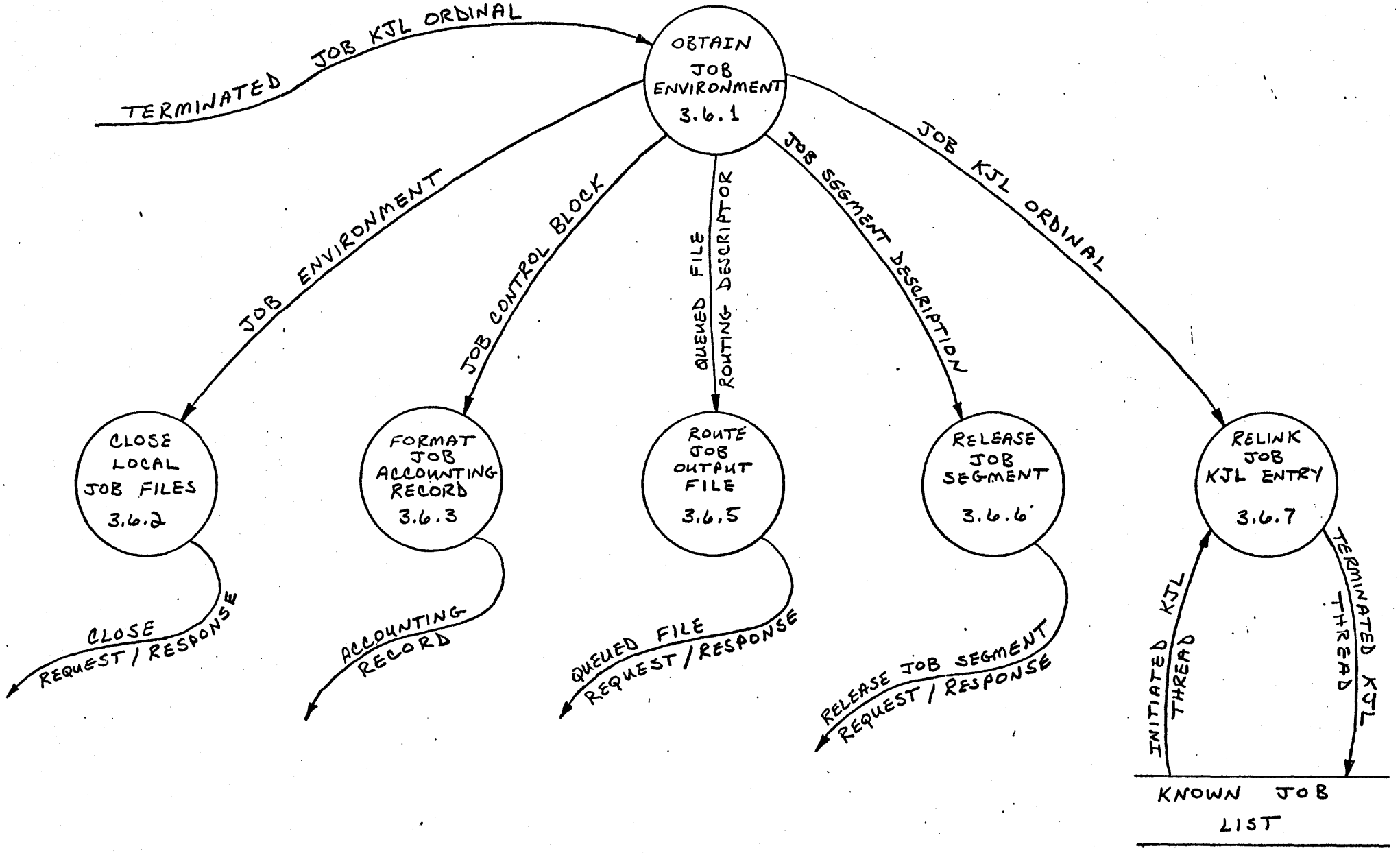
Statement

Command ! assignment ! CONTROL ! ...

Value

- expression .. expression

JOB TERMINATOR
(DIAGRAM 3.6) RAP
02-14-79



```
active_job_status : FLOW
  = io_status
    + task_status
    + memory_assigned;
active_job_status_file : FILE {one entry per active job}
  = <active_job_status>;
aged_job_priority : FLOW
  = job_scheduling_priority
    + class_aging_factor;
alter_request_parameters : FLOW
  = qfile
    + name
    + [priority]
    + [form]
    + [repeat_count]
    + status;
assignment : FLOW
  = variable
    + "="
    + expression;
command : FLOW
  = command_name
    + ["," ]
    + parameter_list;
command_name : FLOW
  = [file_name]
    + "."
    + parameter_list;
```

```
ontrol_statement : FLOW
 = control_statement_name
   + [",",]
   + parameter_list;
ontrol_statement_name : FLOW
 = (if
   ! ifend
   ! cycle
   ! exit
   ! for
   ! forend
   ! goto
   ! loop
   ! loopend
   ! revert);
ivert_request_parameters : FLOW
 = (terminal
   + queue_type
   + destination_user
   + status
   ! owned
   + queue_type
   + destination_user
   + status
   ! family
   + queue_type
   + family
   + destination_family
```

```
    ! qfile
    + name
    + destination_user
    + status);
drop_request_parameters : FLOW
  = (all_terminal
    + queue_type
    + no_output_status
    ! all_owned
    + queue_type
    + no_output
    + status
    ! qfile
    + name
    + no_output
    + status);
foreign_text : FLOW
  = <ascii charcter>
    + (",
    ! ";"
    ! ":"
    ! end_of_line);
highest_priority_job_kill_entry : FLOW
  = input_job_entry;
initiated_kill_thread : FLOW
  = <initiated_job_entries>;
input_job_entry : FLOW
  = user_name
```

OUT 79/02/27. 15.30.36.

```
   + job_name
   + job_mode
   + job_type
   + job_class
   + job_status
   + job_scheduling_priority
   + input_file_catalog_description
   + family_to_execute
   + default_output_destination
   + kjl_thread_linkage;
ob_control_block : FLOW
 = job_identification
   + job_limits
   + job_accounting_data
   + job_control_data;
ob_environment : FLOW
 = job_segment_description
   + job_control_block
   + job_input_file
   + job_kjl_ordinal
   + job_template;
ob_kjl_ordinal : FLOW
 = kjl_ordinal;
ob_mode : FLOW
 = (interactive
   ! batch);
ob_priority : FLOW
 = job_scheduling_priority;
```

```
job_segment_description : FLCW
  = process_virtual_address;
job_state : FLOW
  = (queued
    ! deferred
    ! initiated
    ! terminated);
job_template : FLCW
  = task_services_code_segments
    + task_monitor_code_segments
    + initial_job_control_block
    + initial_execution_control_block
    + global_program_description;
job_template_list : FILE
  = standard_template
    + maintenance_template
    + operator_template;
job_type : FLOW
  = (standard
    ! maintenance
    ! operator);
known_job_list : FILE
  = [<(queued_job_entries
    ! deferred_job_entries
    ! initiated_job_entries
    ! terminated_job_entries)>];
label : FLOW
  = name
```

```
   + ":";
imits_profile : FLOW
 = user_validation_limits;
ine : FLOW
 = (source_line
   ! data_line);
ame : FLOW
 = alpha
   + {{<alpha>
   ! decimal_digit)};
utput_file_entry : FLOW
 = file_name
   + queue_type
   + user_id
   + destination_user
   + family_name
   + external_characteristics
   + queue_priority
   + repeat_count
   + routing_address;
itput_file_list : FILE
 = <output_file_entry>;
arameter : FLOW
 = (parameter_name
   ! parameter_name
   + "="
   + parameter_value
   ! parameter_value);
```

```
parameter_list : FLOW
  = parameter
    + <",",
    + parameter>;
parameter_name : FLOW
  = name;
parameter_value : FLOW
  = (value_list
    ! foreign_text);
qcount_request_parameters : FLOW
  = (terminal
    + [external_characteristics]
    + counts
    + status
    ! owned
    + counts
    + status
    ! family
    + counts
    + status);
qstatus_request_parameters : FLOW
  = (terminal
    + queue_type
    + priority
    + [external_characteristics]
    + start
    + status_array
    + nreturned
```

```
   + status
   ! owned
   + queue_type
   + start
   + status_array
   + nreturned
   + status
   ! family
   + queue_type
   + priority
   + family
   + start
   + status_array
   + nreturned
   + status
   ! file
   + name_array
   + nnames
   + status_array
   + status);
status_type_identification : FLOW
 = (full
   ! brief);
ueued_file_count : FLOW
 = <queue_type
   + file_count>;
ueued_file_request_response : FLOW
 = route_request_response
```

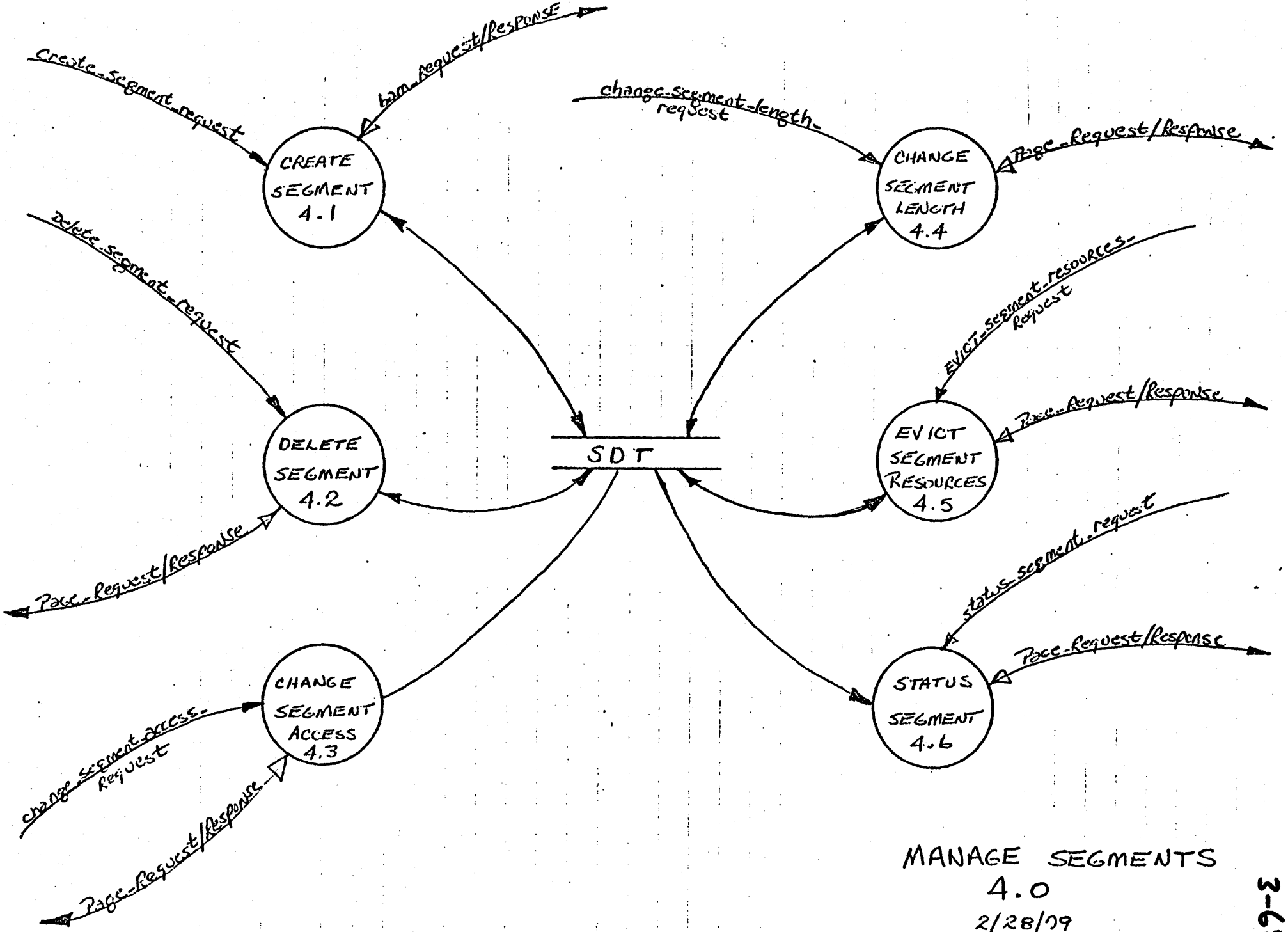
```
    ! qstatus_request_response
    ! qcount_request_response
    ! drop_request_response
    ! divert_request_response
    ! alter_request_response;
queued_file_routing_descriptor : FLOW
  = local_file_name
    + queue_type
    + user_id
    + destination_user
    + queue_reference_name
    + family_name
    + routing_form_code
    + queue_file_priority
    + repeat_count
    + routing_address;
queued_kji_thread : FLOW
  = <queued_job_entries>;
scheduled_job_kji_ordinal : FLOW
  = kji_ordinal;
source_line : FLOW
  = [label]
    + statement_list;
statement : FLOW
  = (command
    ! control_statement
    ! assignment);
statement_list : FLOW
```

OUT 79/02/27. 15.30.36.

```
 = statement
   + < ";"
   + statement>;
tatus_array : FLOW
 = <queue_reference_name
   + queue_type
   + queue_file_priority
   + user_identification>;
erminated_job_kjl_ordinal : FLOW
 = kjl_ordinal;
erminated_kjl_thread : FLOW
 = <terminated_job_entries>;
ser_identification : FLOW
 = username
   + password;
alue : FLOW
 = expression
   + ["."]
   + expression;
alue_list : FLOW
 = (value_set
   ! "("value_set
   + <",",
   + value_set>")");
alue_set : FLOW
 = (value
   ! "("value
   + <",",value>")");
```

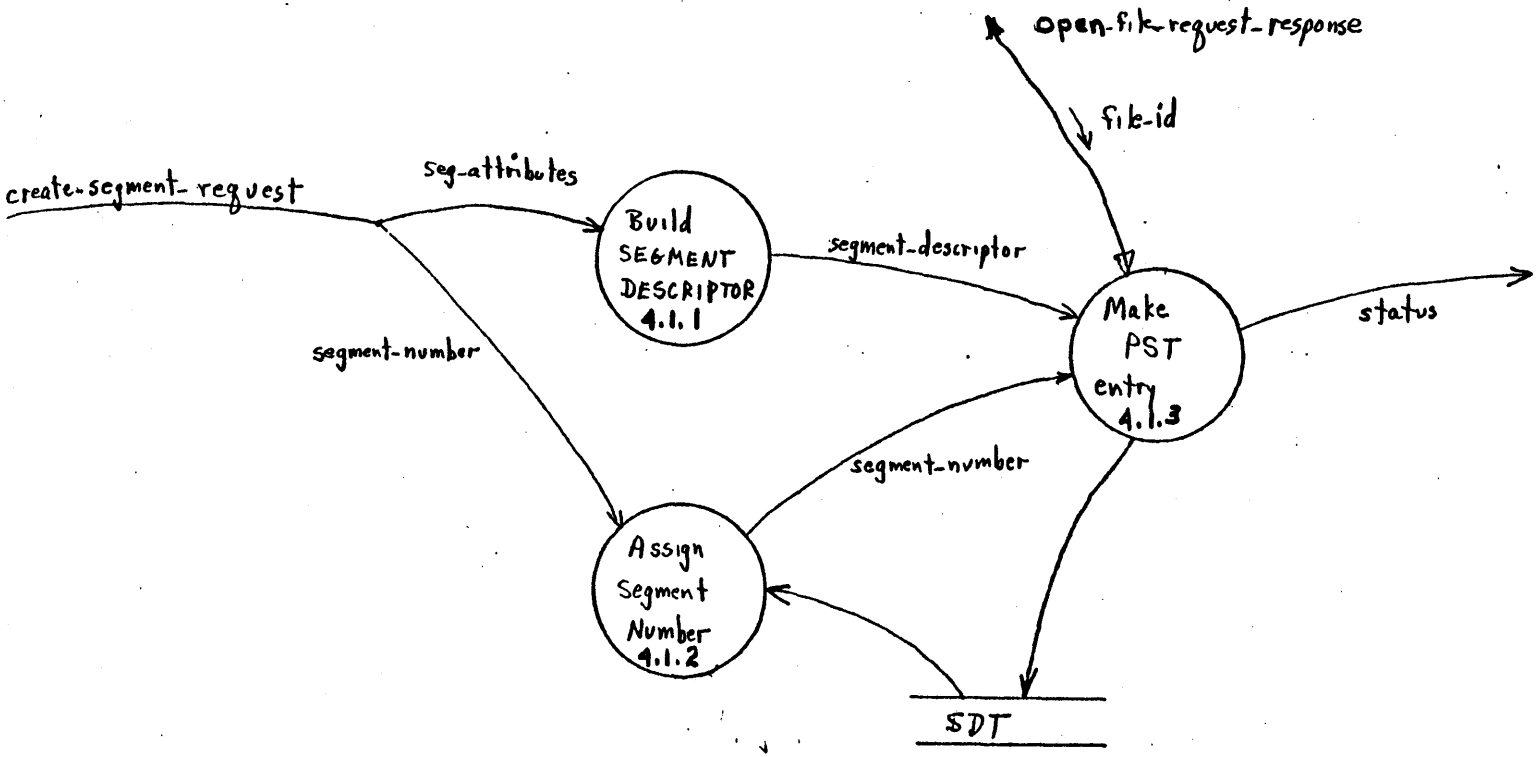
```
variable : FLOW
  = name
    + ['('
    + expressior')']
    + [',',
    + name
    + ['('expression')'];
```

3.4 MANAGE SEGMENTS

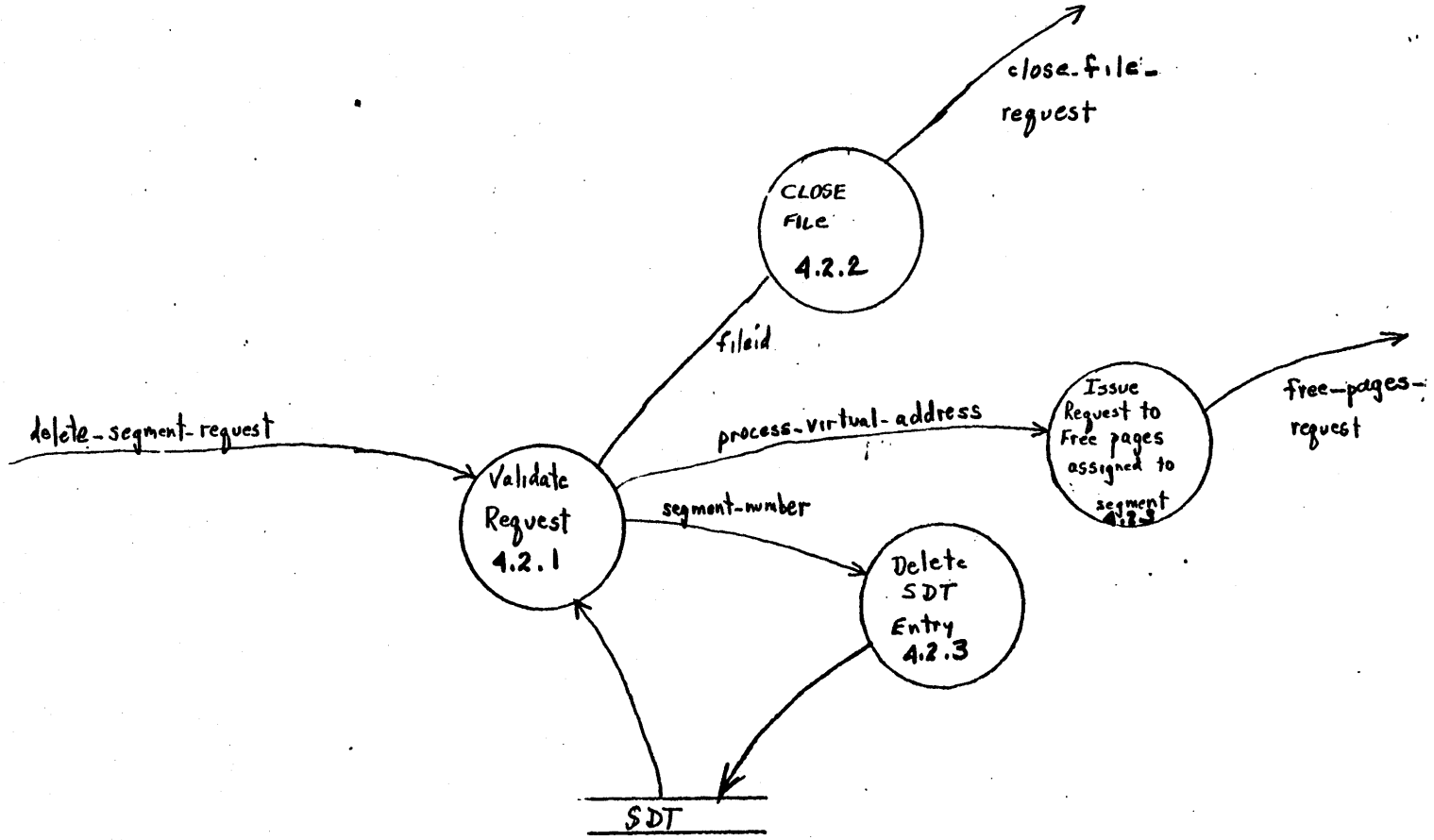


MANAGE SEGMENTS
4.0
2/28/79

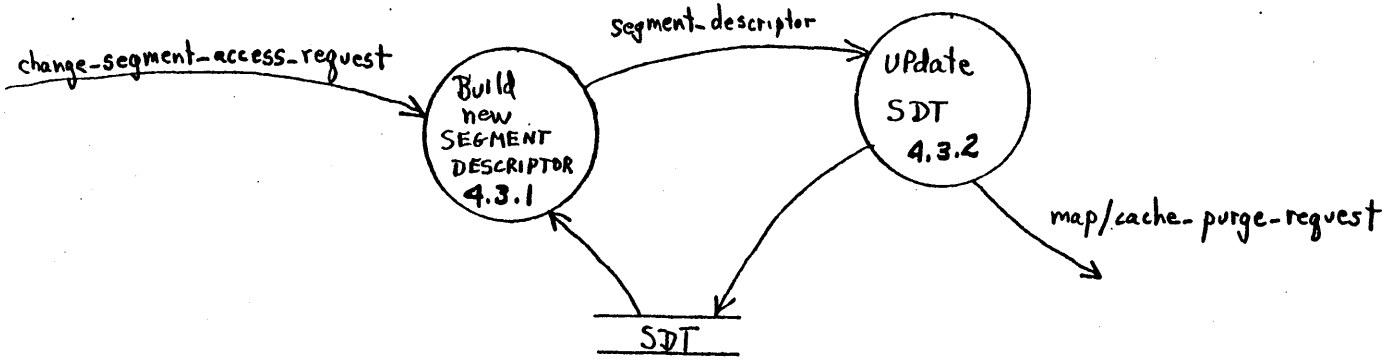
89-2



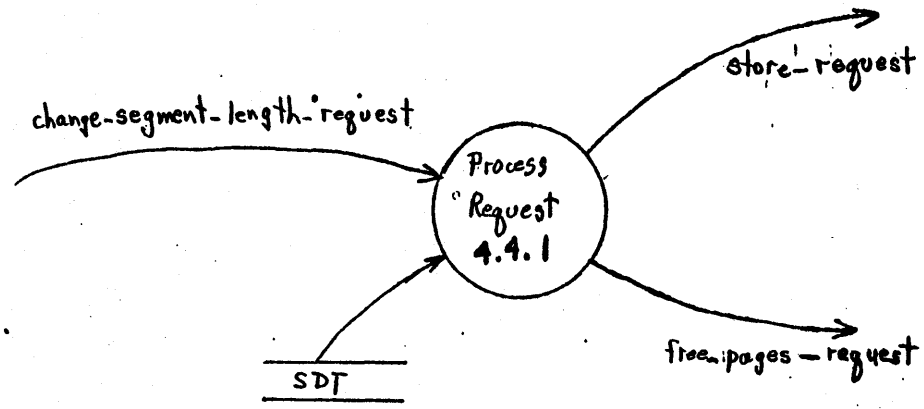
4.1 CREATE_SEGMENT Request
2/28/79

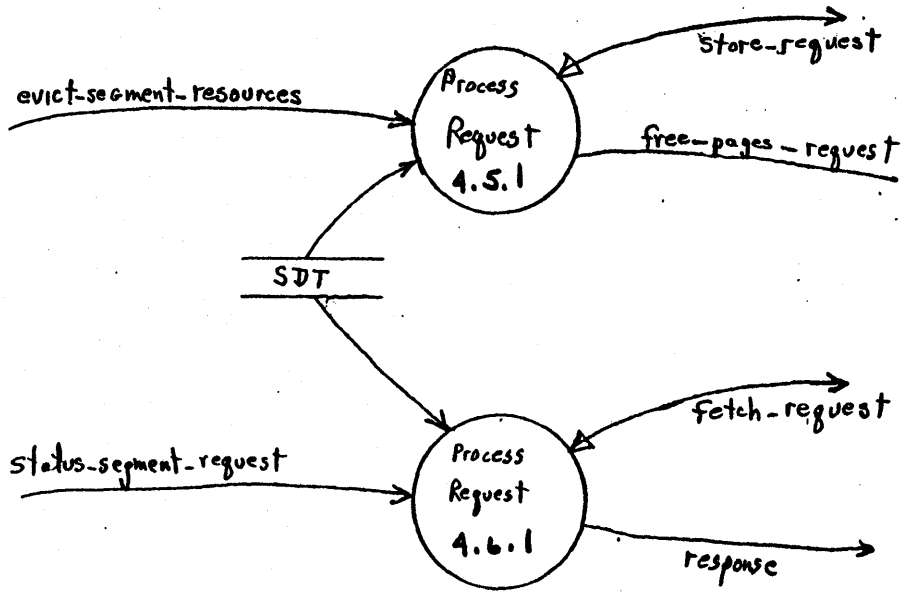


4.2 DELETE SEGMENT Request
2/28/79



4.3 CHANGE_SEGMENT_ACCESS Request
2/28/79





4.4 CHANGE_SEGMENT_LENGTH request
4.5 EVICT_SEGMENT_RESOURCES request
4.6 STATUS_SEGMENT request

2/28/79

OUT 79/02/26. 08.42.40.

```
advise_in_request : FLOW
  = process_virtual_address
    + length;
advise_out_request : FLOW
  = process_virtual_address
    + length
    + (wait
    ! nowait);
assign_page_request : FLOW
  = free_page_request;
change_segment_access_request : FLOW
  = process_virtual_address
    + segment_attributes;
change_segment_length_request : FLOW
  = process_virtual_address
    + maximum_segment_length;
create_segment_request : FLOW
  = segment_attribute
    + segment_number
    + segment_length;
delete_segment_request : FLOW
  = process_virtual_address;
evict_segment_resources_request : FLOW
  = process_virtual_address;
existing_page : FLOW
  memory_descriptor;
free_page_request : FLOW
  = process_virtual_address
```

OUT 79/02/26. 08.42.40.

```
    + length;
lock_page_request : FLOW
  = free_page_request;
memory_descriptor : FLOW
  = system_virtual_address
    + taskid
    + number_of_pages;
new_page : FLOW
  = memory_descriptor;
page_descriptor : FLOW
  = system_virtual_address
    + taskid
    + page_frame_table_index;
page_frame_table : FILE
  = <page_age
    + page_table_index
    + active_io_count
    + time_stamp
    + queued_taskid
    + queue_type
    + running_job_ordinal>;
page_id : FILE
  = active_segment_id
    + page_offset;
page_table : FILE
  = <page_id
    + physical_address
    + [valid]
```

```
    + [continue]
    + [used]
    + [modified]>;
process_virtual_address : FLOW
  = ring_number
    + segment_number
    + byte_offset;
queue_type : FILE
  = (free
    ! available_modified
    ! shared_working_set
    ! wired
    ! available
    ! job_working_set);
segment_attributes : FLOW
  = tbd "gives information required to set up a segment descriptor entry";
segment_descriptor_table : FILE
  = <[wired]
    + [shared]
    + [stack]
    + [sequential]
    + [cache_by_pass]
    + [read]
    + [write]
    + [binary]
    + [execute]
    + [local_key]
    + [global_key]
```

```
    + file_id
    + active_segment_id>;
segment_number : FLOW
  = 0..4095;
status_segment_request : FLOW
  = process_virtual_address;
status_segment_response : FLOW
  = maximum_segment_length
    + current_segment_length
    + segment_attributes;
swap_physical_request : FLOW
  = running_job_id;
swap_physical_response : FLOW
  = <real_memory_address
    + system_virtual_address>
    + working_set_size;
swap_working_set_request : FLOW
  = running_job_id;
unlock_page : FLOW
  = free_page_request;
```

3.5 MANAGE PROGRAMS

CDC DATA FLOW DIAGRAM WORKSHEET

Author: Date: Page of

Note:

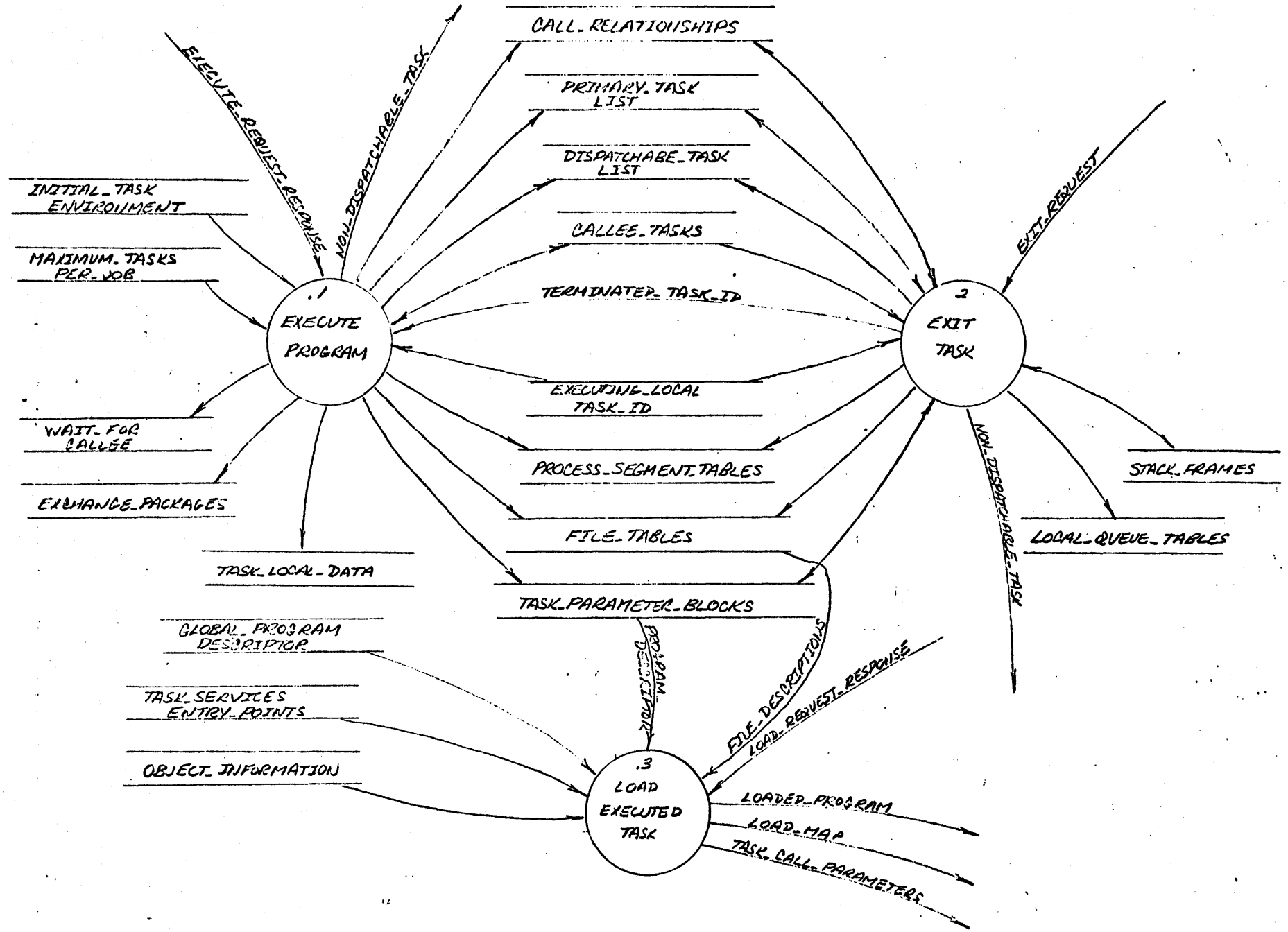
Diagram No: Rev: Diagram Name: Superior Diagram Name:

CDC DATA FLOW DIAGRAM WORKSHEET

Author: L. E. LESHNER Date: 7/28/79 Page 1 of 1

Note:

Diagram No: 5.1 Rev: Diagram Name: MANAGING TASKS Superior Diagram Name: MANAGE PROGRAMS


CALL_RELATIONSHIPS
PRIMARY_TASK LIST
DISPATCHABE_TASK LIST
CALLEE_TASKS
TERMINATED_TASK_ID
EXECUTING_LOCAL TASK_ID
PROCESS_SEGMENT_TABLES
FILE_TABLES
TASK_PARAMETER_BLOCKS
EXECUTE_REQUEST_RESPONSE
NON_DISPATCHABLE_TASK
INITIAL_TASK ENVIROUMENT
MAXIMUM_TASKS PER_JOB
WAIT_FOR CALLEE
EXCHANGE_PACKAGES
TASK_LOCAL_DATA
GLOBAL_PROGRAM DESCRIPTOR
TASK_SERVICES ENTRY_POINTS
OBJECT_INFORMATION
PROGRAM_DESCRIPTORS
FILE_DESCRIPTIONS
LOAD_REQUEST_RESPONSE
LOADED_PROGRAM
LOAD_MAP
TASK_CALL_PARAMETERS
EXIT_REQUEST
NON_DISPATCHABLE_TASK
STACK_FRAMES
LOCAL_QUEUE_TABLES
.1 EXECUTE PROGRAM
.2 EXIT TASK
.3 LOAD EXECUTED TASK

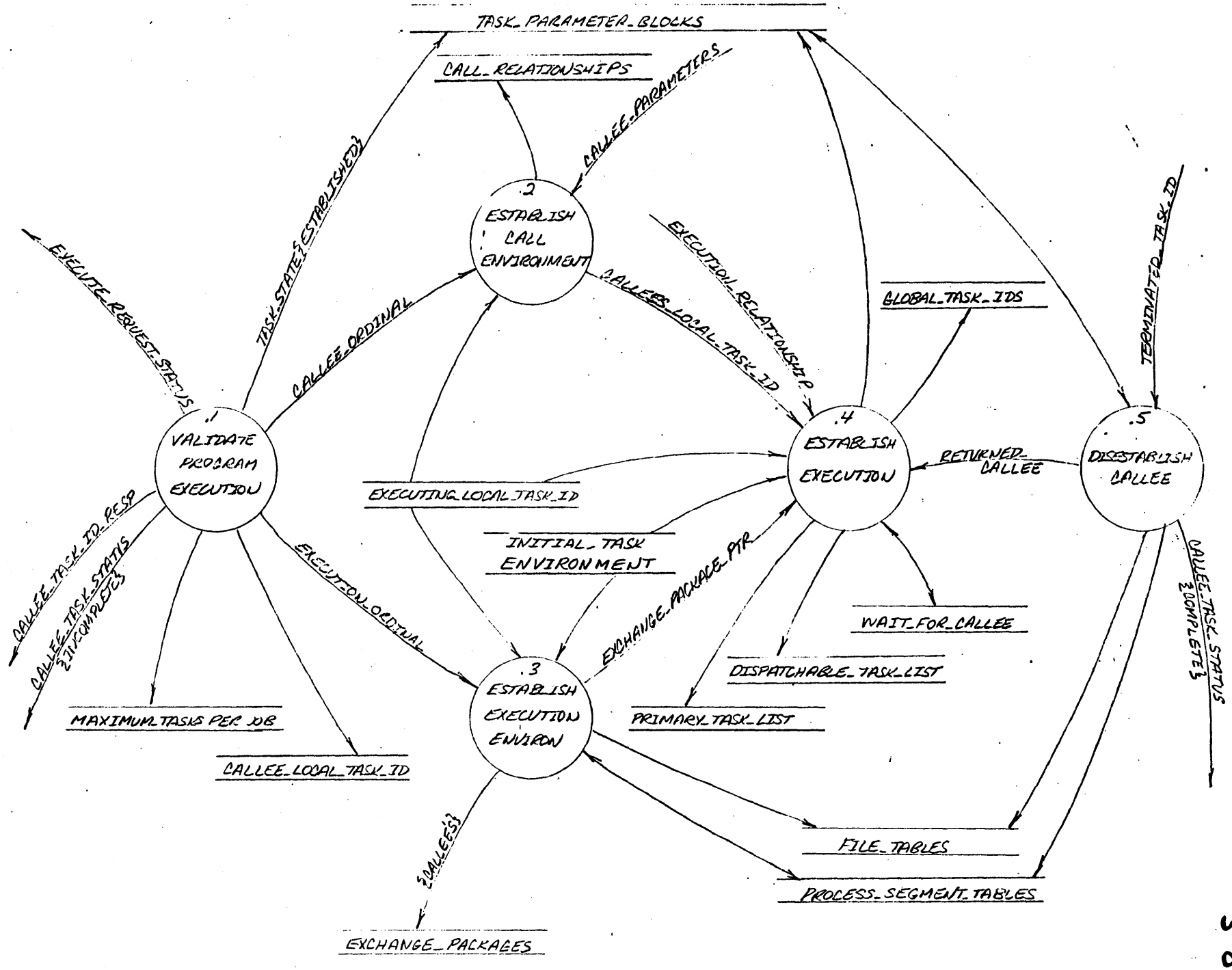


Diagram No:

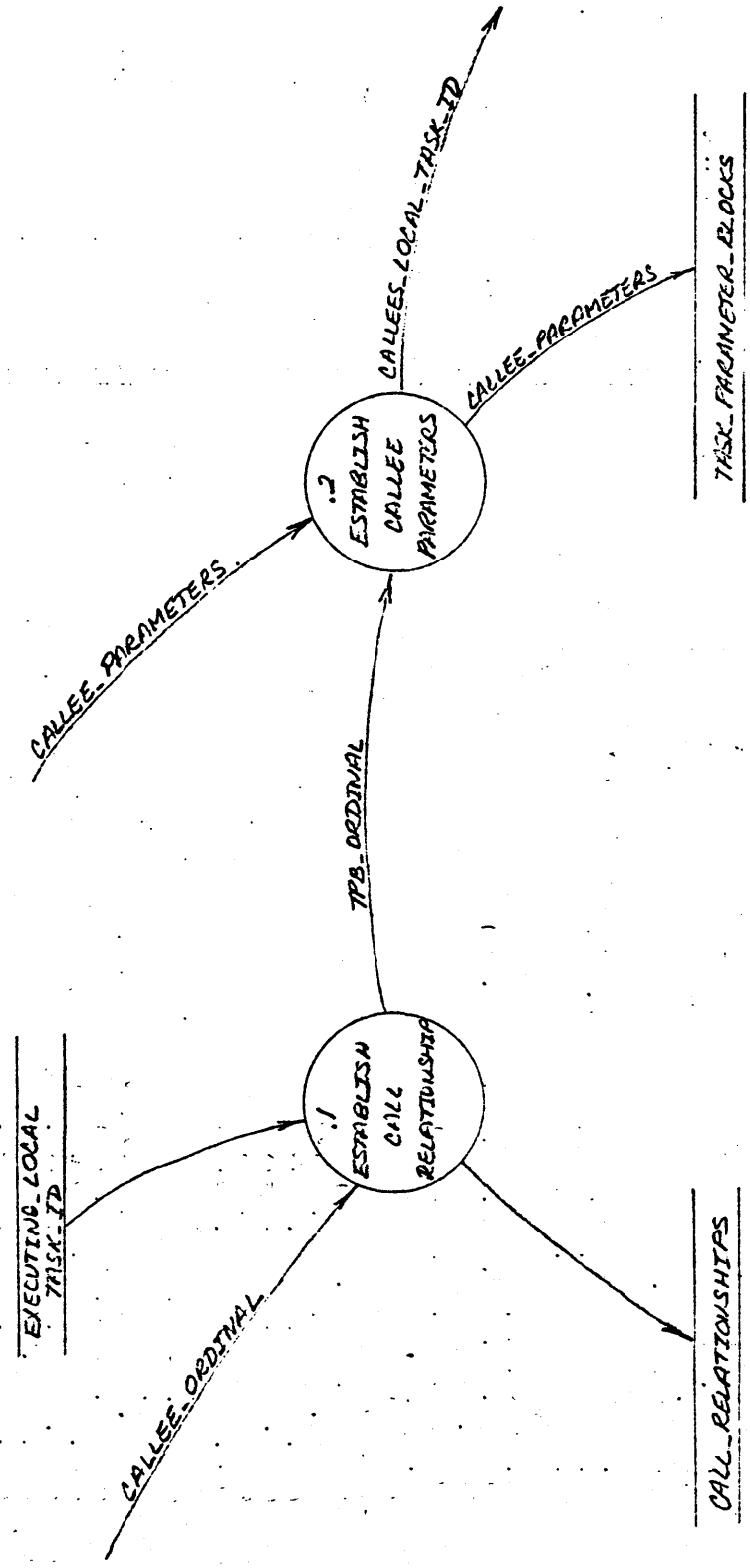
Rev:

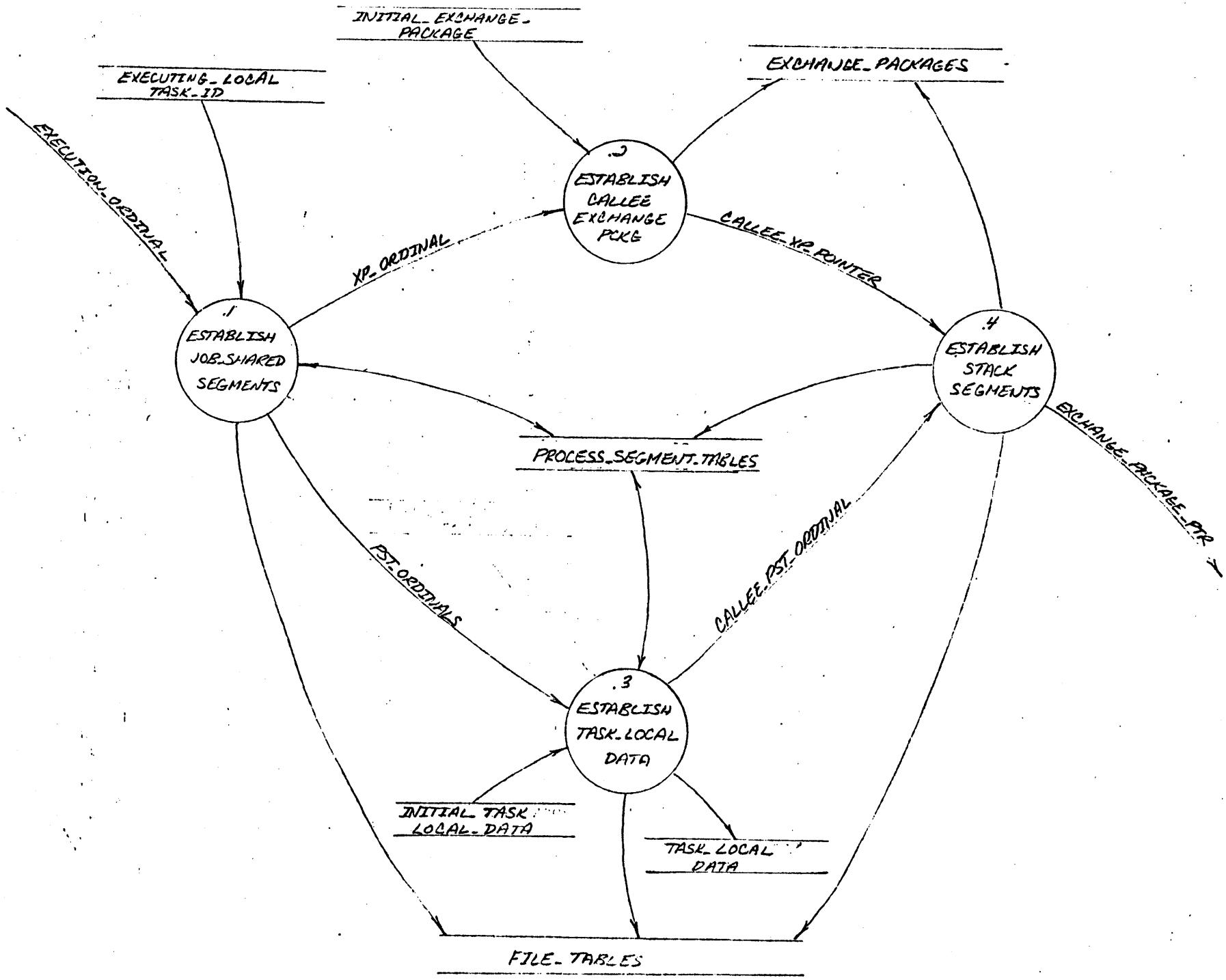
Diagram Name:

Superior Diagram Traffic

3-81

5.1.1.1.2 ESTABLISH CALL ENVIRONMENT

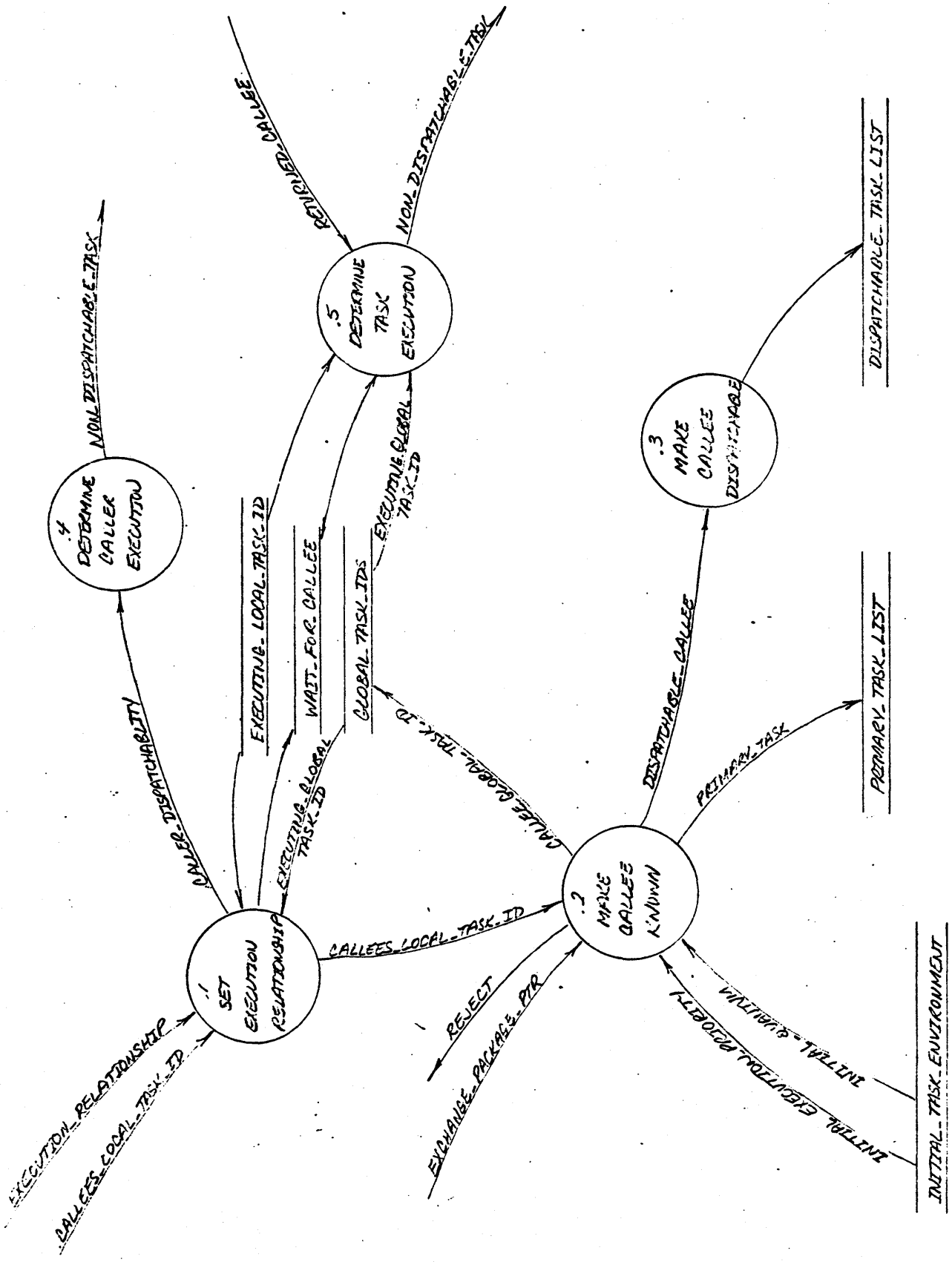


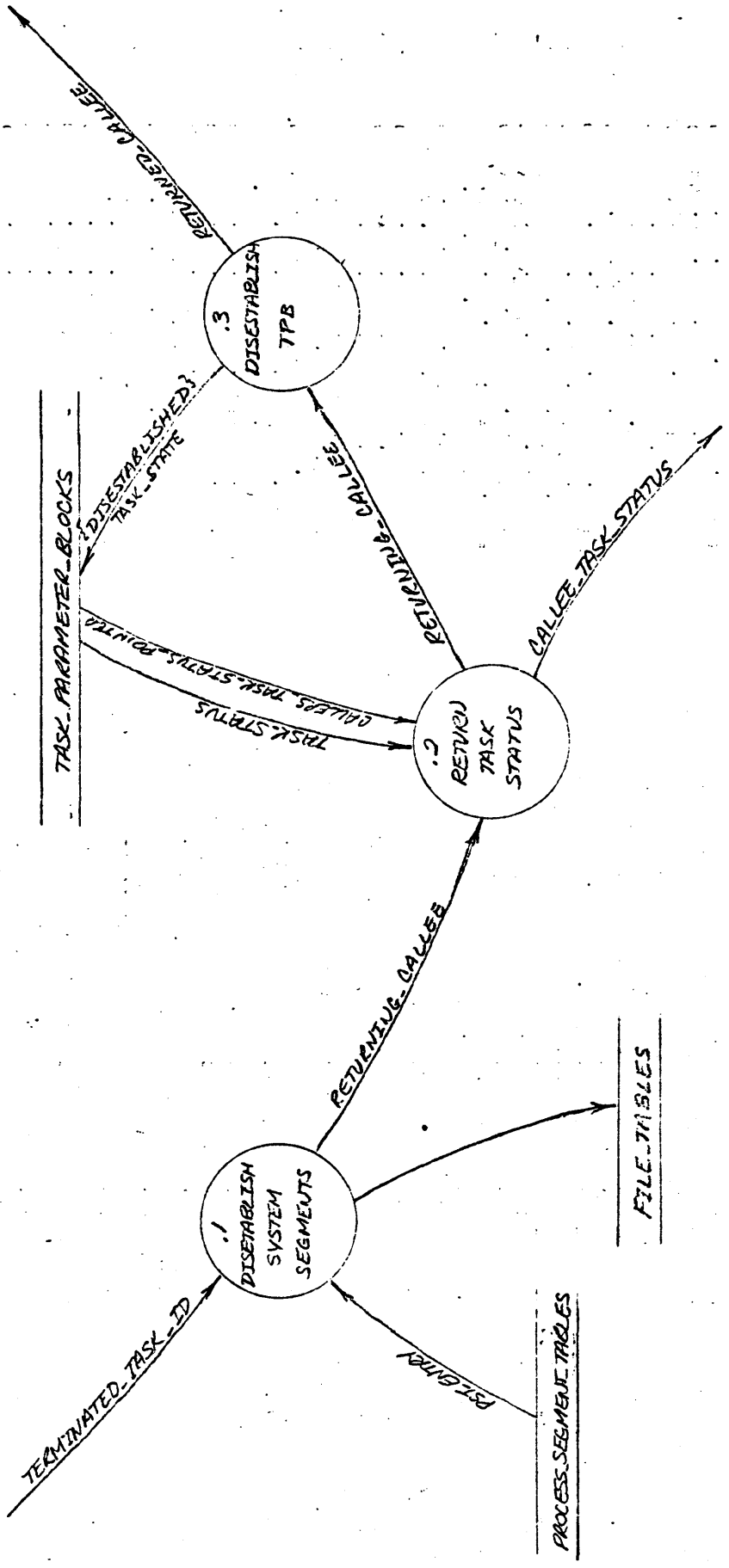

INITIAL_EXCHANGE_PACKAGE
EXCHANGE_PACKAGES
EXECUTING_LOCAL TASK_ID
EXECUTION_ORDINAL
.2 ESTABLISH CALLEE EXCHANGE PCKG
XP_ORDINAL
CALLEE_XP_POINTER
.1 ESTABLISH JOB_SHARED SEGMENTS
.4 ESTABLISH STACK SEGMENTS
EXCHANGE_PACKAGE_PTR
PROCESS_SEGMENT_TABLES
PST_ORDINALS
CALLEE_PST_ORDINAL
.3 ESTABLISH TASK_LOCAL DATA
INITIAL_TASK LOCAL_DATA
TASK_LOCAL DATA
FILE_TABLES

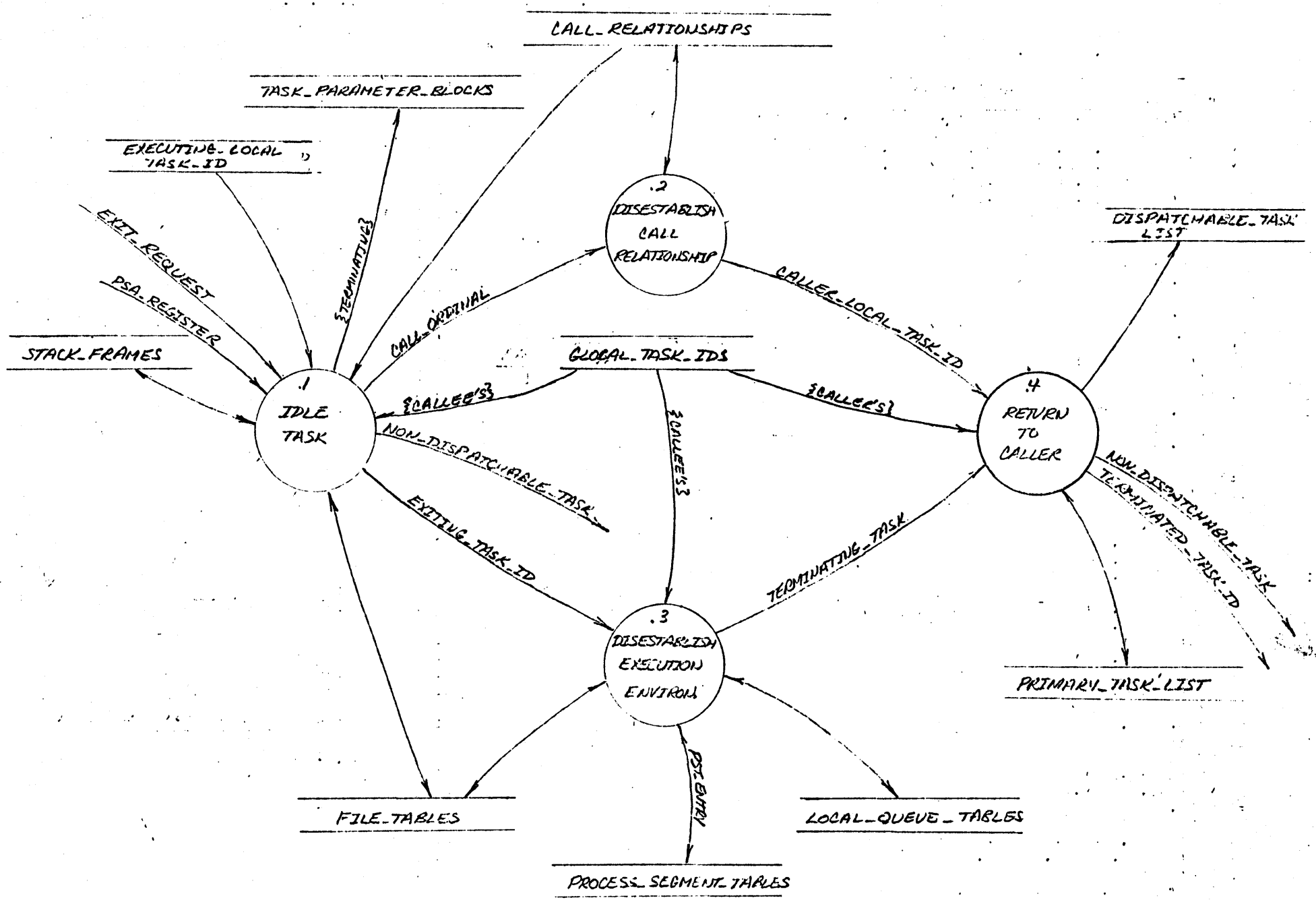
CDC DATA FLOW DIAGRAM WORKSHEET

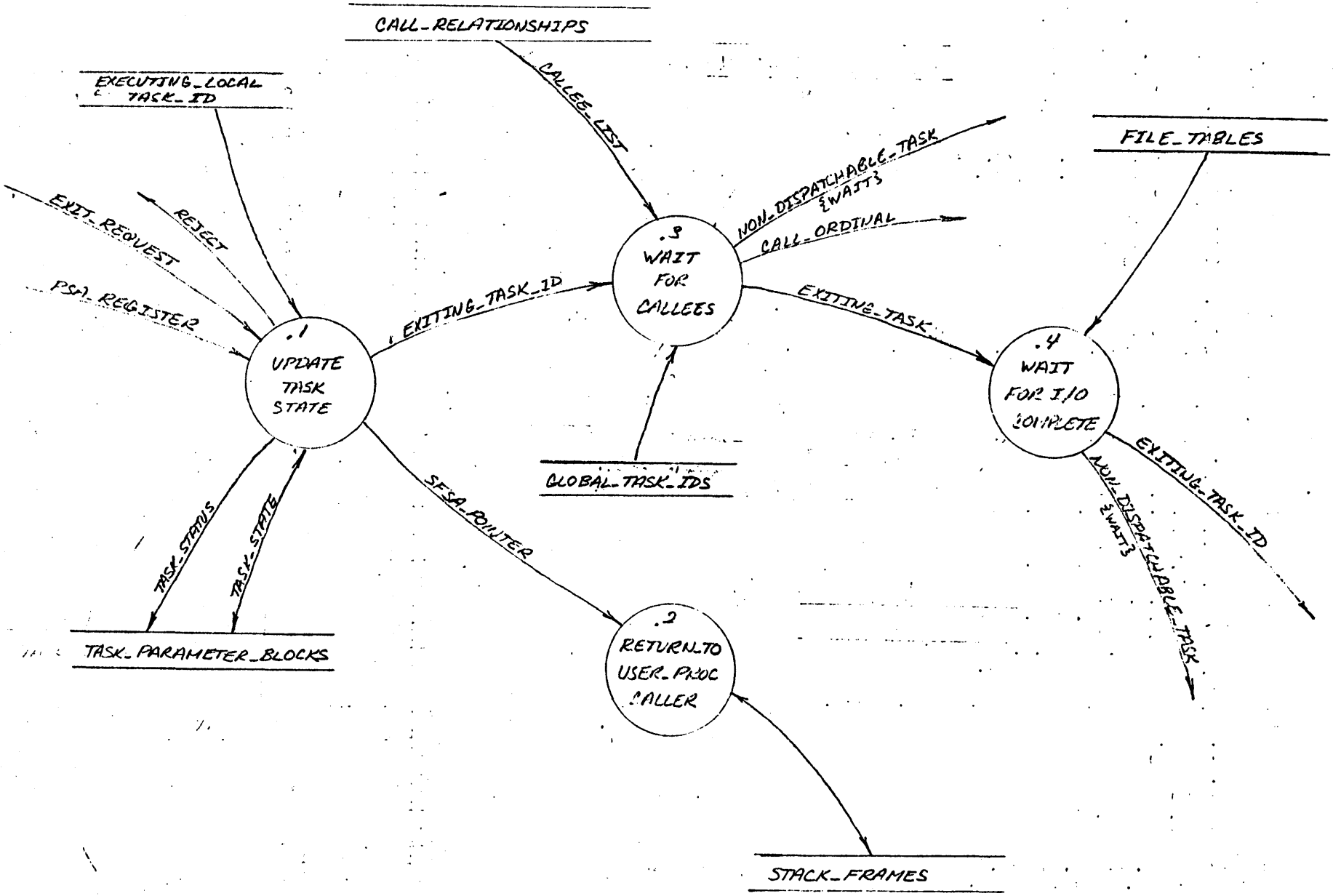
Author:	L. E. LECKMEN	Date:	2/26/79	Page	1 of 1
Note:					
Diagram No:	T.1.1.1.4	Rev:	Diagram Name:	ESTABLISH EXECUTION	Superior Diagram Name: EXECUTE PROGRAM

3-84

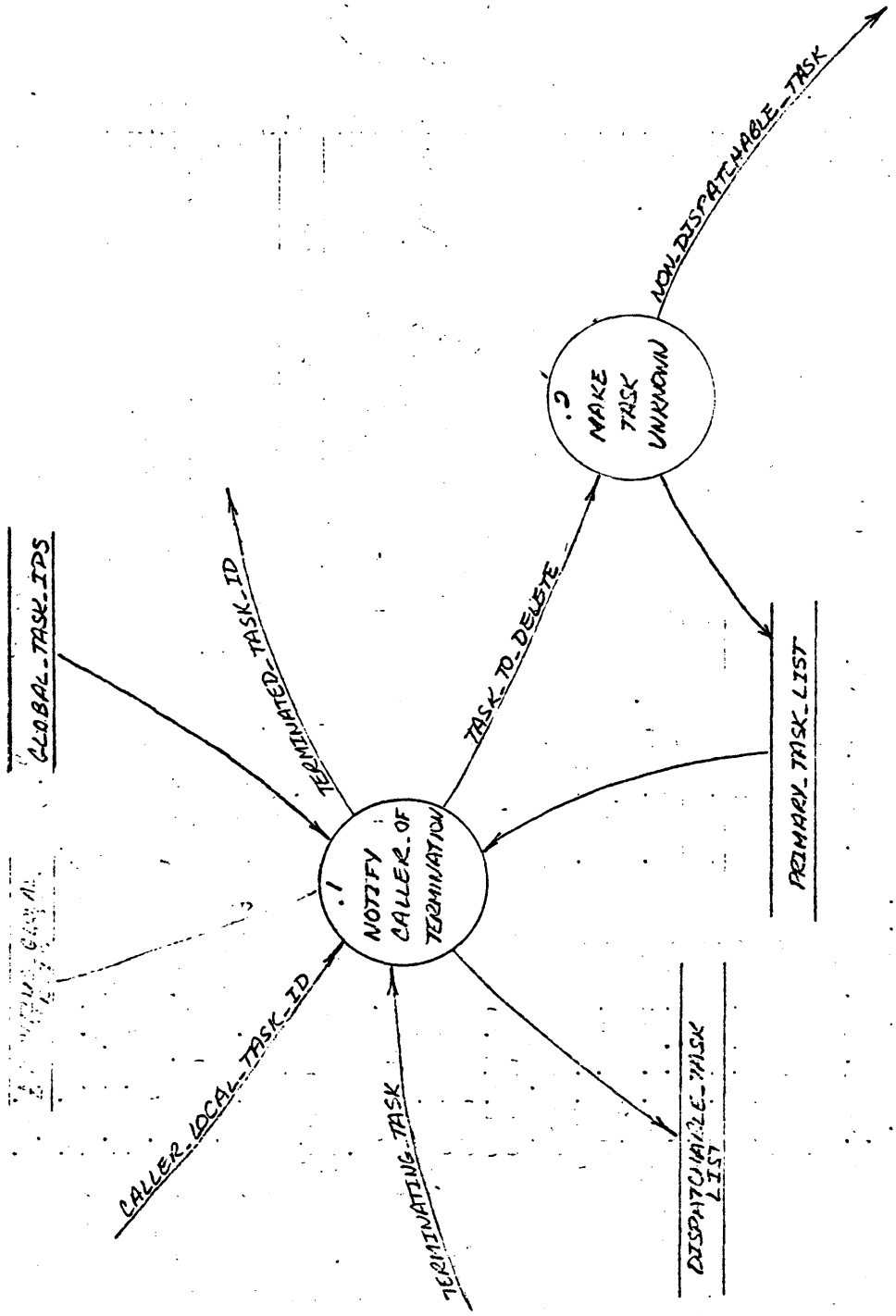



RETURNED_CALLEE
.3 DISESTABLISH TPB
TASK_PARAMETER_BLOCKS
DISESTABLISHED TASK_STATE
RETURNED_CALLEE
CALLEE_TASK_STATUS
CALLEES TASK_STATUS_POINTER
TASK_STATUS
.2 RETURN TASK STATUS
RETURNING_CALLEE
.1 DISESTABLISH SYSTEM SEGMENTS
FILE_TABLES
TERMINATED_TASK_ID
RETURN
PROCESS_SEGMENT_TABLES

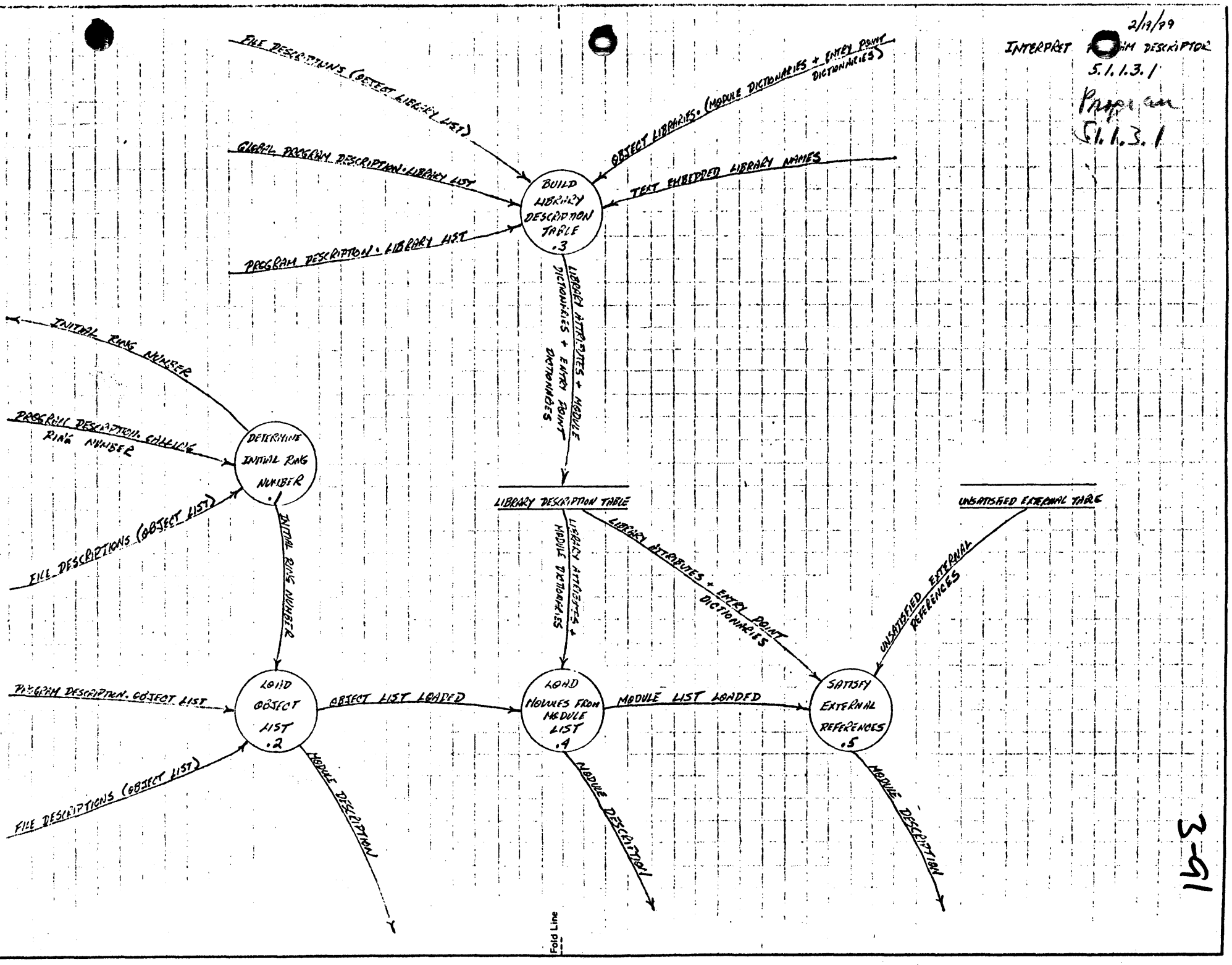

CALL_RELATIONSHIPS
TASK_PARAMETER_BLOCKS
EXECUTING_LOCAL_TASK_ID
EXIT_REQUEST
PSA_REGISTER
STACK_FRAMES
.1 IDLE TASK
{TERMINATIVES}
CALL_ORDINAL
.2 DISESTABLISH CALL RELATIONSHIP
CALLER_LOCAL_TASK_ID
DISPATCHABLE_TASK_LIST
GLOBAL_TASK_IDS
{CALLEE'S}
{CALLEE'S}
{CALLEE'S}
.4 RETURN TO CALLER
NON_DISPATCHABLE_TASK
EXITING_TASK_ID
TERMINATING_TASK
NON_DISPATCHABLE_TASK
TERMINATED_TASK_ID
.3 DISESTABLISH EXECUTION ENVIRON
PRIMARY_TASK_LIST
FILE_TABLES
PDE_ENTRY
LOCAL_QUEUE_TABLES
PROCESS_SEGMENT_TABLES

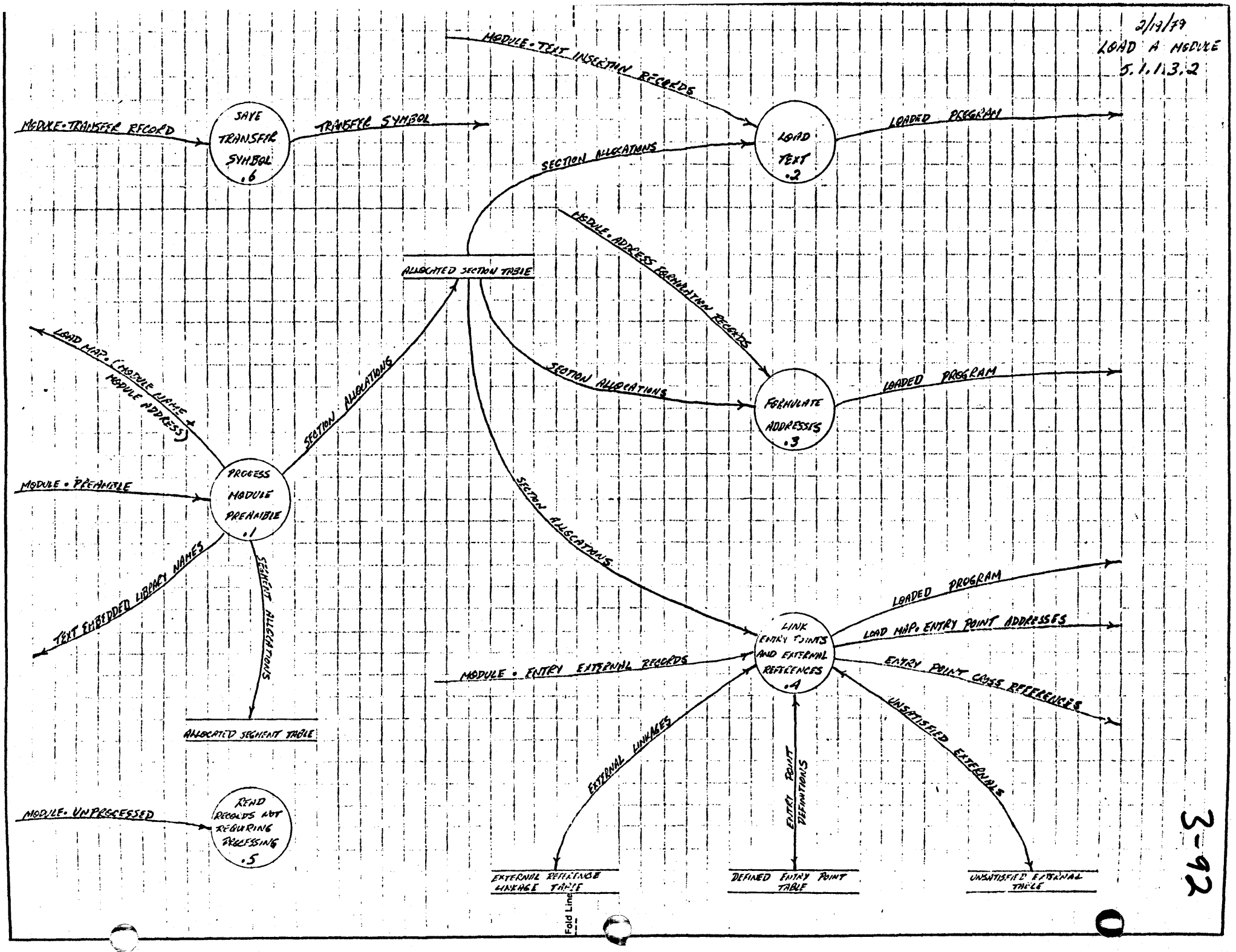

CALL_RELATIONSHIPS
EXECUTING_LOCAL TASK_ID
CALLEE_LIST
FILE_TABLES
EXIT_REQUEST
REJECT
PSW_REGISTER
.1 UPDATE TASK STATE
EXITING_TASK_ID
.3 WAIT FOR CALLEES
NON_DISPATCHABLE_TASK &WAIT3
CALL_ORDINAL
EXITING_TASK
.4 WAIT FOR I/O COMPLETE
GLOBAL_TASK_IDS
TASK_STATUS
TASK_STATE
SFSA_POINTER
.2 RETURN_TO USER_PROC CALLER
DISPATCHABLE_TASK &WAIT3
EXITING_TASK_ID
TASK_PARAMETER_BLOCKS
STACK_FRAMES

.1
DISESTABLISH USER SEGMENTS
.2
CLOSE USER FILES
.3
DISCONNECT FROM QUEUES
EXITING_TASK_ID
EXITING_TASK_ID
EXITING_TASK
TERMINATING_TASK
PROCESS_SEGMENT_TABLES
GLOBAL_TASK_IDS
LOCAL_QUEUE_TABLES
FILE_TABLES


GLOBAL_TASK_IDS
TERMINATED_TASK_ID
TASK_TO_DELETE
NON_DISPATCHABLE_TASK
.2 MAKE TASK UNKNOWN
PRIMARY_TASK_LIST
.1 NOTIFY CALLER OF TERMINATION
CALLER_LOCAL_TASK_ID
TERMINATING_TASK
DISPATCHABLE_TASK LIST

3-90
CDC DATA FLOW DIAGRAM WORKSHEET
Author: Gene Prosser
Date: 3/22/79
Page 1 of 1
Note:
Diagram No: 5.1.1.3
Rev:
Diagram Name: LOAD EXECUTED TASK
Superior Diagram Name:
INTERPRET PROGRAM DESCRIPTION .1
LOAD A MODULE .2
ACCUMULATE ENTRY POINT CROSS REFERENCES .3
WRITE LOAD MAP .5
ESTABLISH TRANSFER ADDRESS .6
DETERMINE ENTRY POINT ADDRESS .8
CREATE STACK SEGMENTS .9
TRANSFER TO USER PROGRAM .7
LOAD MAP
PROGRAM DESCRIPTION
TASK CALL PARAMETERS
TRANSFER ADDRESS
TRANSFER SYMBOL
LOAD MAP. TRANSFER ADDRESS
LOAD MAP. CROSS REFERENCES
MODULE DESCRIPTION
ENTRY POINT DEFINITIONS
TRANSFER SYMBOL
DEFINED ENTRY POINT TABLE
TRANSFER SYMBOL DEFINITION
TASK SERVICES ENTRY POINTS
ENTRY POINT DEFINITIONS
LIBRARY DESCRIPTION TABLE
ENTRY POINT DICTIONARIES
PHYSICAL ADDRESSES
INITIAL RING NUMBER
PHYSICAL ADDRESSES & DICTIONARIES
PROGRAM DESCRIPTION
FILE DESCRIPTION
TASK DESCRIPTION (ENTRY POINT DICTIONARY)
UNSATISFIED EXTERNAL REFERENCES
UNSATISFIED EXTERNAL TABLE
TEXT EMBEDDED LIBRARY NAMES
MODULE DESCRIPTIONS
RINGS LOADED
RINGS LOADED
LOADED PROGRAM
LOADED TASK CALL PARAMETERS
PROGRAM DESCRIPTION (LOAD MAP NAME & LOAD MAP BOTTOM)
LOADED PROGRAM
ALLOCATED SEGMENT TABLE
LOAD MAP. SEGMENT ALLOCATIONS
SEGMENT ALLOCATIONS
MODULES


2/13/79
INTERPRET PROGRAM DESCRIPTOR
5.1.1.3.1
Program
5.1.1.3.1
FILE DESCRIPTIONS (OBJECT LIBRARY LIST)
GLOBAL PROGRAM DESCRIPTION • LIBRARY LIST
PROGRAM DESCRIPTION • LIBRARY LIST
OBJECT LIBRARIES • (MODULE DICTIONARIES + ENTRY POINT DICTIONARIES)
TEXT EMBEDDED LIBRARY NAMES
BUILD LIBRARY DESCRIPTION TABLE .3
LIBRARY ATTRIBUTES + MODULE DICTIONARIES + ENTRY POINT DICTIONARIES
INITIAL RING NUMBER
PROGRAM DESCRIPTION • CALLING RING NUMBER
FILE DESCRIPTIONS (OBJECT LIST)
DETERMINE INITIAL RING NUMBER .1
INITIAL RING NUMBER
LIBRARY DESCRIPTION TABLE
LIBRARY ATTRIBUTES + MODULE DICTIONARIES
LIBRARY ATTRIBUTES + ENTRY POINT DICTIONARIES
UNSATISFIED EXTERNAL THERE
UNSATISFIED EXTERNAL REFERENCES
PROGRAM DESCRIPTION. OBJECT LIST
FILE DESCRIPTIONS (OBJECT LIST)
LOAD OBJECT LIST .2
OBJECT LIST LOADED
LOAD MODULES FROM MODULE LIST .4
MODULE LIST LOADED
SATISFY EXTERNAL REFERENCES .5
MODULE DESCRIPTION
MODULE DESCRIPTION
MODULE DESCRIPTION
Fold Line
3-91

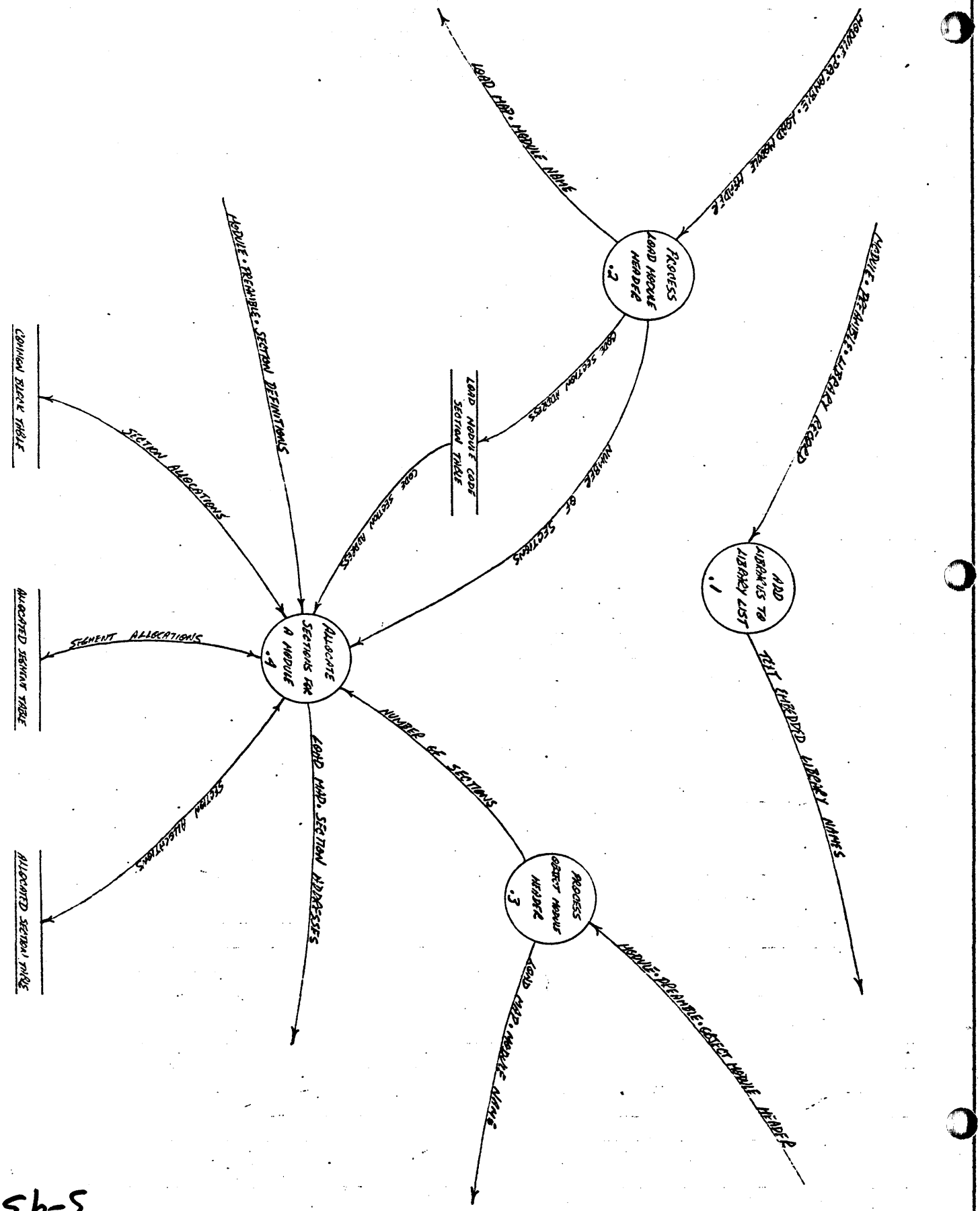

2/19/79
LOAD A MODULE
5.1.1.3.2
MODULE • TEXT INSERTION RECORDS
MODULE • TRANSFER RECORD
SAVE TRANSFER SYMBOL .6
TRANSFER SYMBOL
LOAD TEXT .2
LOADED PROGRAM
SECTION ALLOCATIONS
ALLOCATED SECTION TABLE
MODULE • ADDRESS FORMATION RECORDS
LOAD MAP • (MODULE NAME + MODULE ADDRESS)
SECTION ALLOCATIONS
FORMULATE ADDRESSES .3
LOADED PROGRAM
MODULE • PREAMBLE
PROCESS MODULE PREAMBLE .1
SECTION ALLOCATIONS
SEGMENT ALLOCATIONS
TEXT EMBEDDED LIBRARY NAMES
LINK ENTRY POINTS AND EXTERNAL REFERENCES .4
LOADED PROGRAM
LOAD MAP • ENTRY POINT ADDRESSES
ENTRY POINT CROSS REFERENCES
MODULE • ENTRY EXTERNAL RECORDS
UNSATISFIED EXTERNALS
ALLOCATED SEGMENT TABLE
EXTERNAL LINKAGES
ENTRY POINT DEFINITIONS
MODULE • UNPROCESSED
READ RECORDS NOT REQUIRING PROCESSING .5
EXTERNAL REFERENCE LINKAGE TABLE
DEFINED ENTRY POINT TABLE
UNSATISFIED EXTERNAL TABLE
Fold Line
3-92

CDC DATA FLOW DIAGRAM WORKSHEET

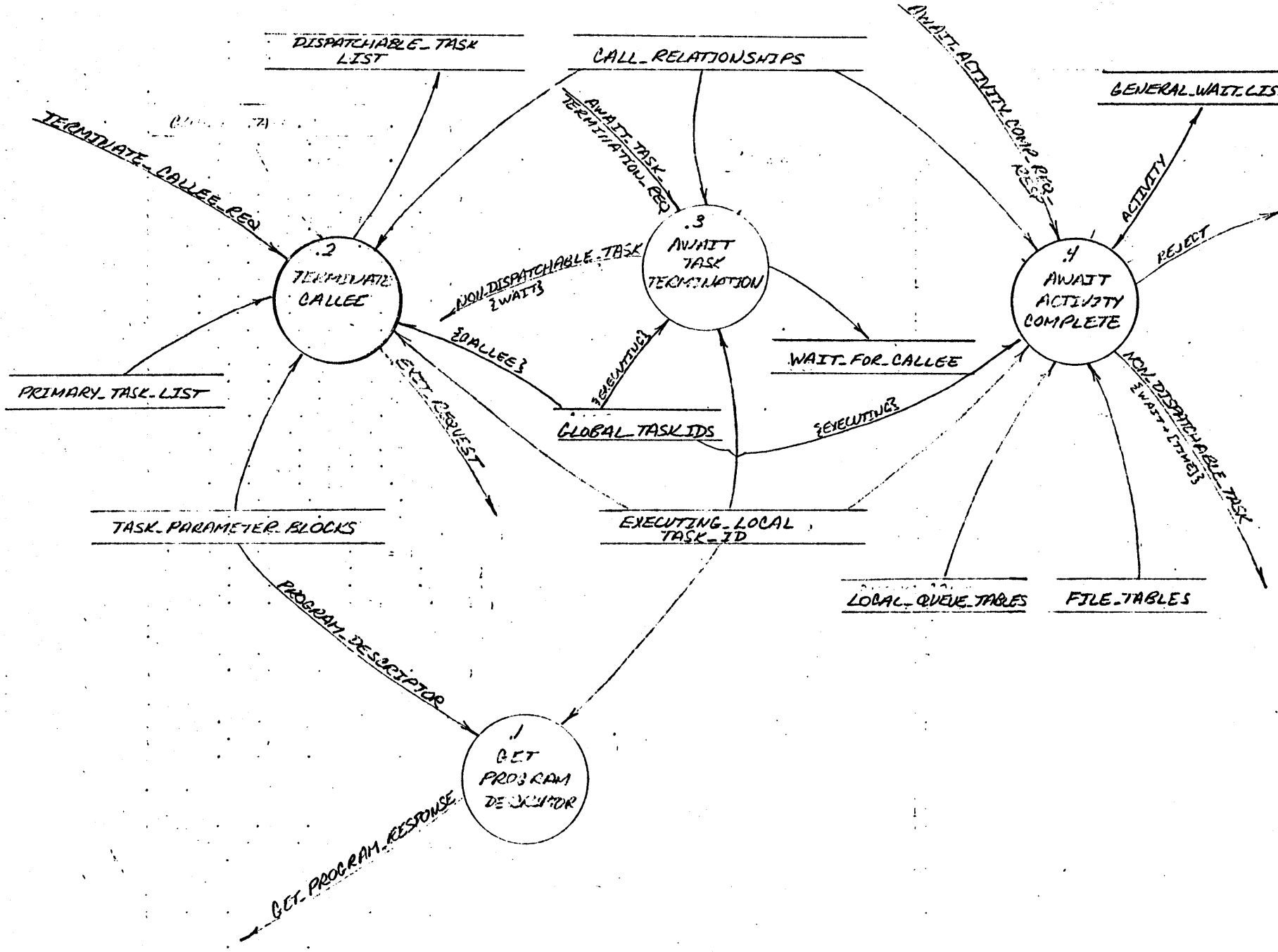
Author: GRACE HOPLETT Date: 2/27/79 Page 1 of 1

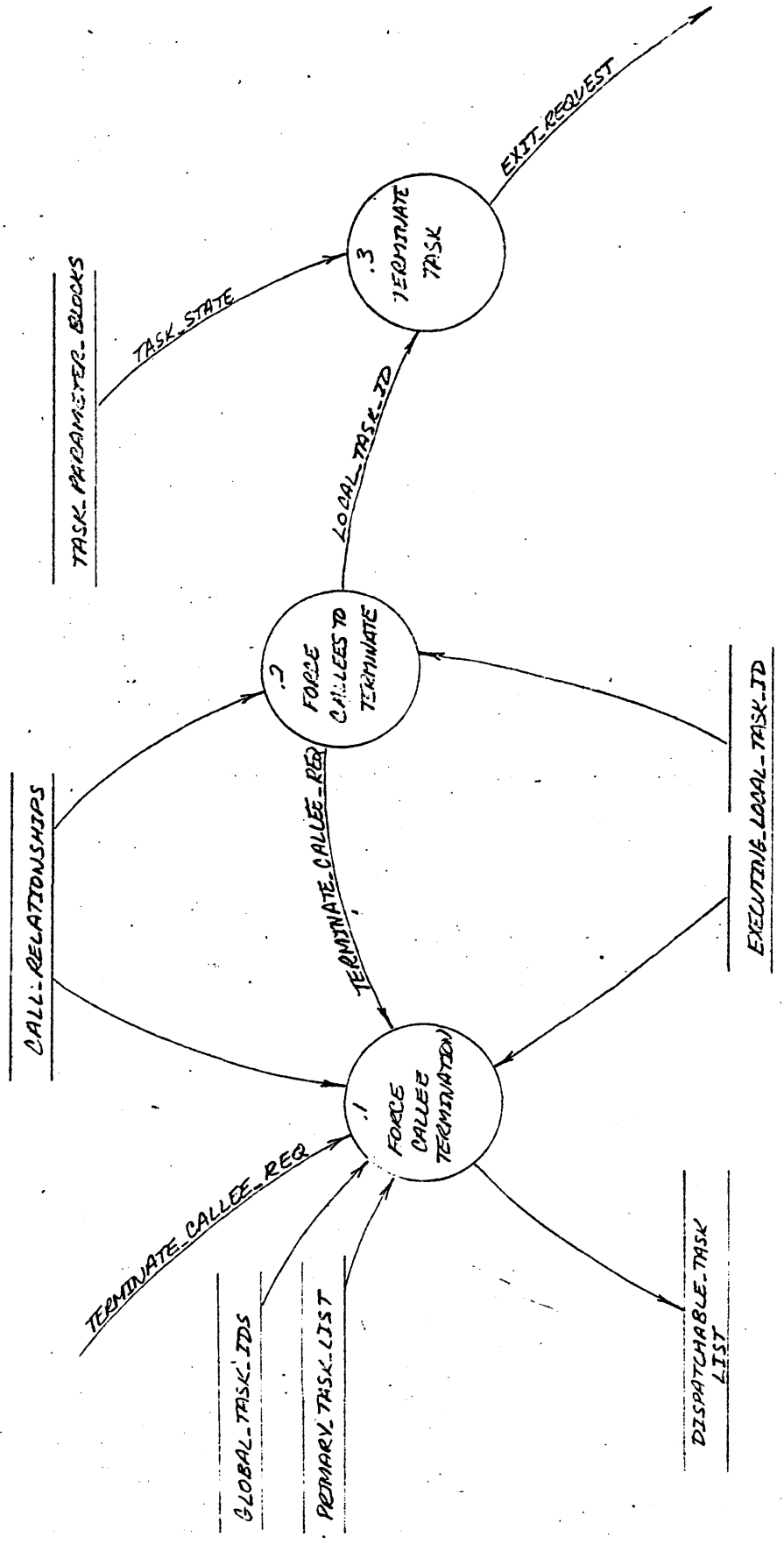
Note:

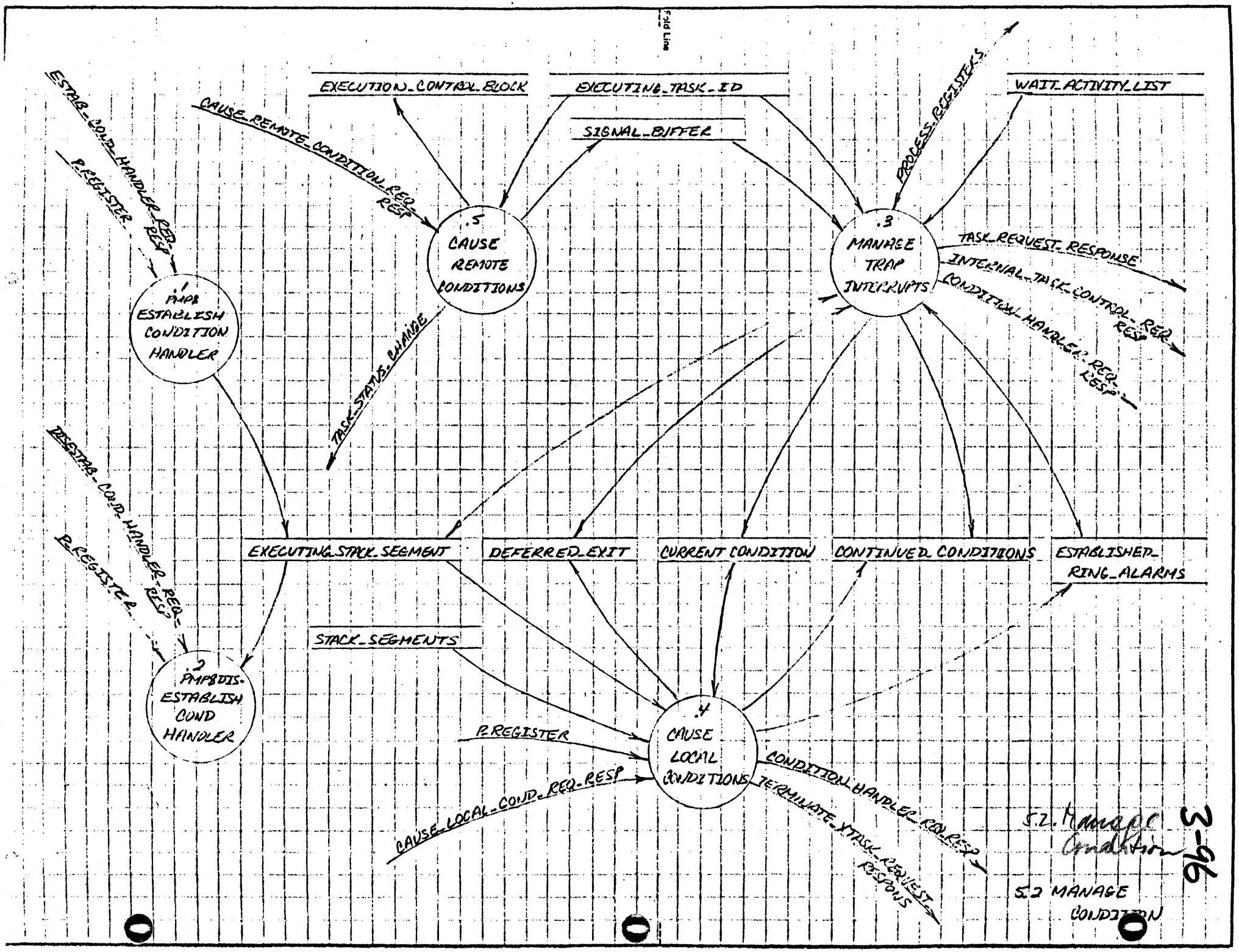
Diagram No: 5.1.1.3.2.1 Rev: Diagram Name: PROCESS MODULE TREE FILES Superior Diagram Name:



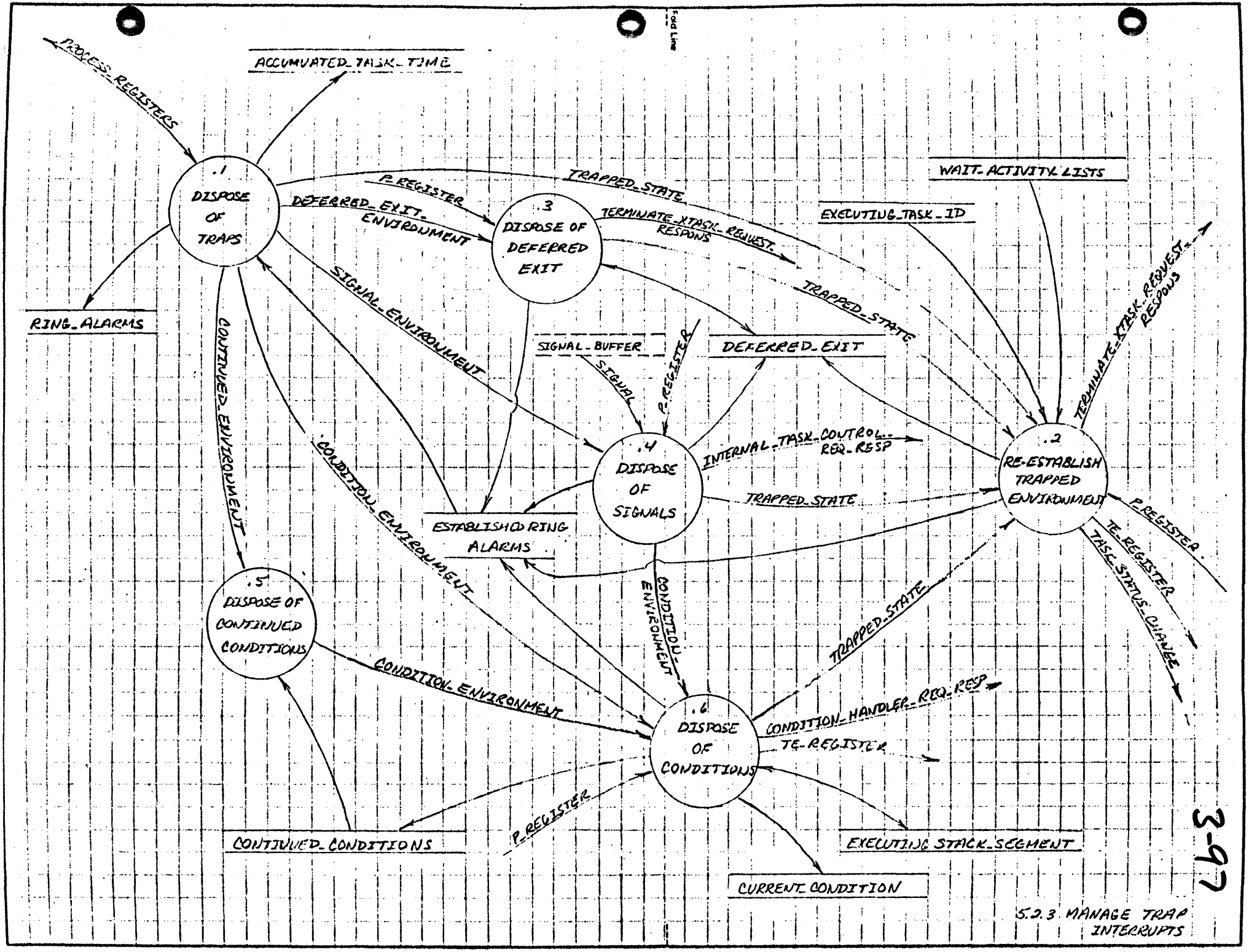
3-23

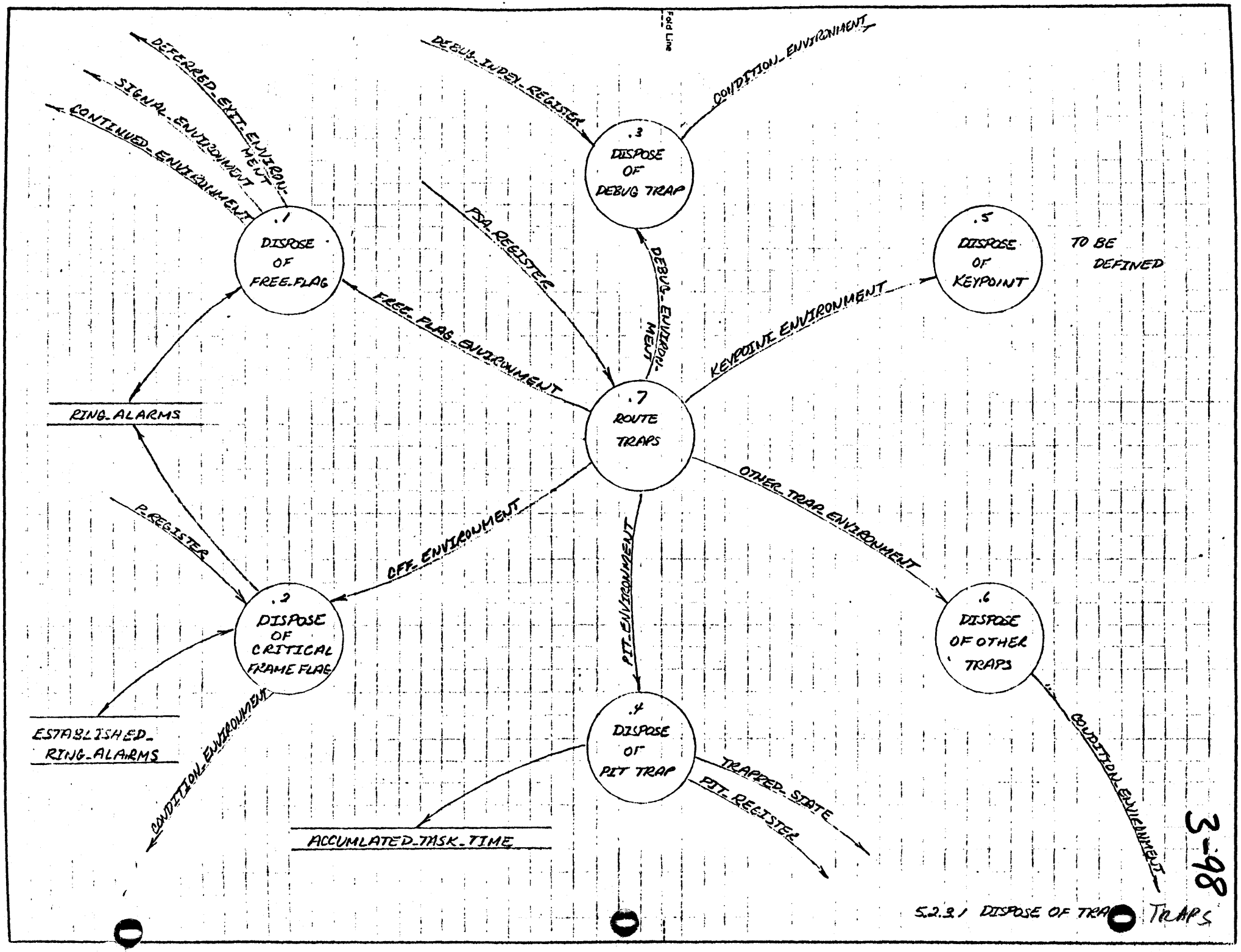

DISPATCHABLE_TASK LIST
CALL_RELATIONSHIPS
AWAIT_ACTIVITY_COMP_REQ/RESP
GENERAL_WAIT_LIST
TERMINATE_CALLEE_REQ
AWAIT_TASK_TERMINATION_REQ
ACTIVITY
.2 TERMINATE CALLEE
.3 AWAIT TASK TERMINATION
.4 AWAIT ACTIVITY COMPLETE
REJECT
NON_DISPATCHABLE_TASK {WAIT}
{CALLEE}
{EXECUTING}
WAIT_FOR_CALLEE
PRIMARY_TASK_LIST
EXIT_REQUEST
GLOBAL_TASK_IDS
{EXECUTING}
ACN_DISPATCHABLE_TASK {WAIT, EXIT}
TASK_PARAMETER_BLOCKS
EXECUTING_LOCAL TASK_ID
LOCAL_QUEUE_TABLES
FILE_TABLES
PROGRAM_DESCRIPTOR
.1 GET PROGRAM DESCRIPTOR
GET_PROGRAM_RESPONSE

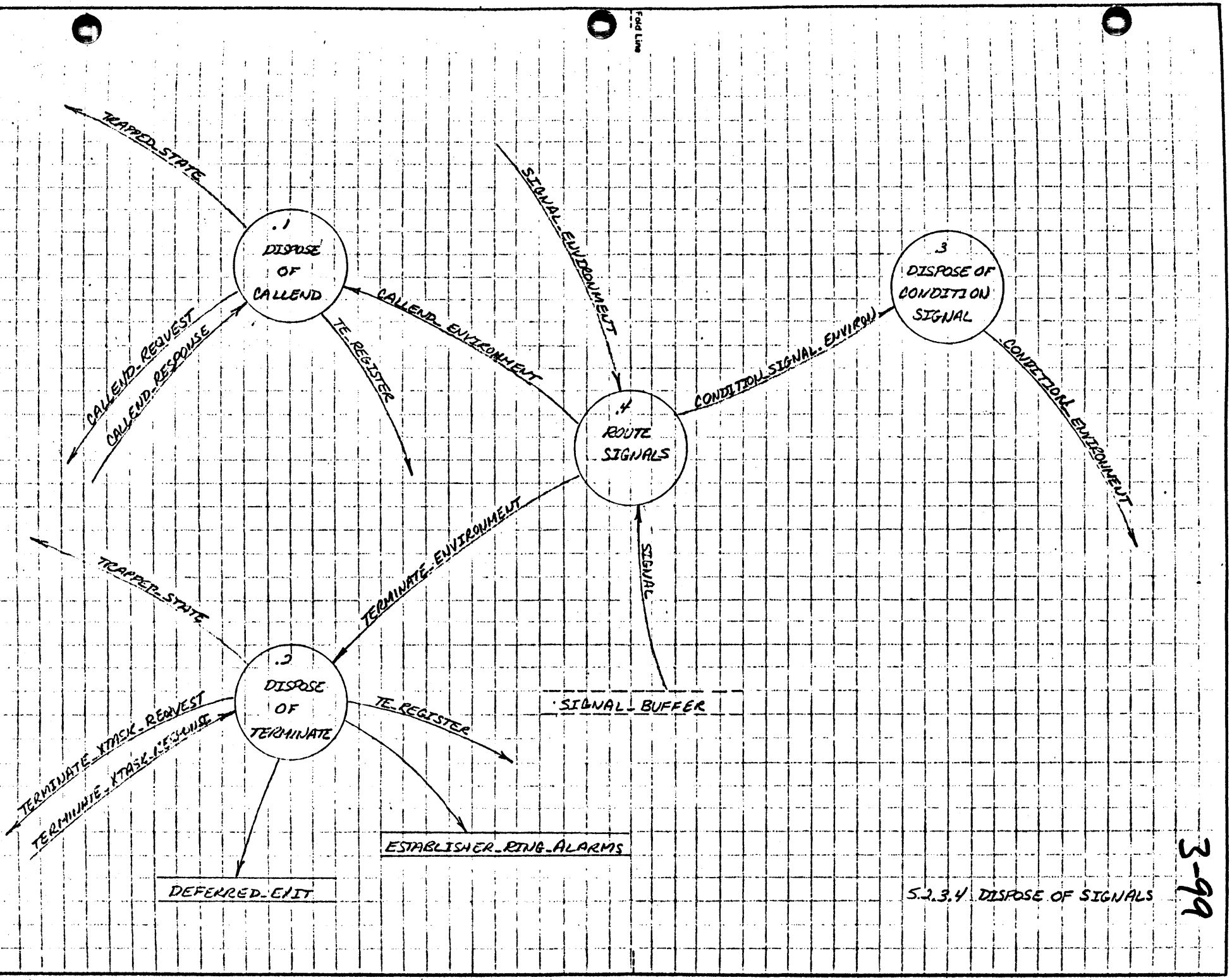

.1
FORCE CALLEE TERMINATION
.2
FORCE CALLEES TO TERMINATE
.3
TERMINATE TASK
TERMINATE_CALLEE_REQ
CALL_RELATIONSHIPS
TERMINATE_CALLEE_REQ
GLOBAL_TASK_IDS
PRIMARY_TASK_LIST
TASK_PARAMETER_BLOCKS
TASK_STATE
LOCAL_TASK_ID
EXIT_REQUEST
EXECUTING_LOCAL_TASK_ID
DISPATCHABLE_TASK_LIST


Fold Line
EXECUTION_CONTROL_BLOCK
EXECUTING_TASK_ID
WAIT_ACTIVITY_LIST
SIGNAL_BUFFER
PROCESS_REGISTERS
ESTAB_COND_HANDLER_REQ-RESP
P_REGISTER
CAUSE_REMOTE_CONDITION_REQ-RESP
.5 CAUSE REMOTE CONDITIONS
.3 MANAGE TRAP INTERRUPTS
TASK_REQUEST_RESPONSE
INTERNAL_TASK_CONTROL_REQ-RESP
CONDITION_HANDLER_REQ-RESP
1 PMP8 ESTABLISH CONDITION HANDLER
TASK_STATUS_CHANGE
DESTAB_COND_HANDLER_REQ-RESP
P_REGISTER
EXECUTING_STACK_SEGMENT
DEFERRED_EXIT
CURRENT CONDITION
CONTINUED CONDITIONS
ESTABLISHED_RING_ALARMS
STACK_SEGMENTS
2 PMP8DIS-ESTABLISH COND HANDLER
.4 CAUSE LOCAL CONDITIONS
P_REGISTER
CONDITION_HANDLER_REQ-RESP
CAUSE_LOCAL_COND_REQ-RESP
TERMINATE_XTASK_REQUEST_RESPONS
5.2 MANAGE CONDITION

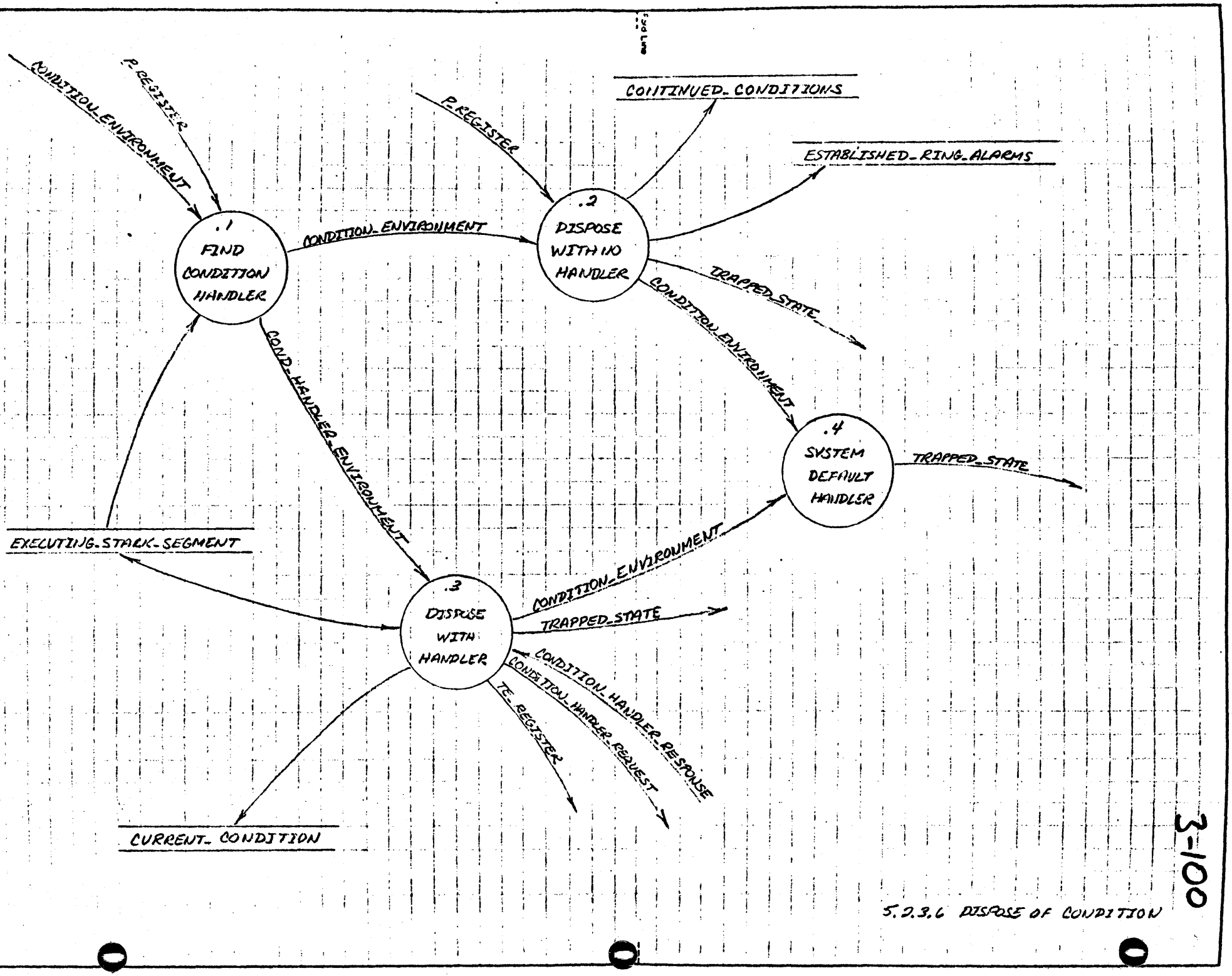
3-96


PROCESS_REGISTERS
ACCUMULATED_TASK_TIME
.1 DISPOSE OF TRAPS
P_REGISTER
TRAPPED_STATE
DEFERRED_EXIT_ENVIRONMENT
.3 DISPOSE OF DEFERRED EXIT
TERMINATE_XTASK_REQUEST_RESPONS
WAIT_ACTIVITY_LISTS
EXECUTING_TASK_ID
RING_ALARMS
SIGNAL_ENVIRONMENT
TRAPPED_STATE
SIGNAL_BUFFER
SIGNAL
P_REGISTER
DEFERRED_EXIT
TERMINATE_XTASK_REQUEST_RESPONS
CONTINUED_ENVIRONMENT
.4 DISPOSE OF SIGNALS
INTERNAL_TASK_CONTROL_REQ_RESP
.2 RE-ESTABLISH TRAPPED ENVIRONMENT
CONDITION_ENVIRONMENT
TRAPPED_STATE
ESTABLISHED RING ALARMS
P_REGISTER
TE_REGISTER
TASK_STATUS_CHANGE
.5 DISPOSE OF CONTINUED CONDITIONS
CONDITION_ENVIRONMENT
TRAPPED_STATE
CONDITION_HANDLER_REQ_RESP
.6 DISPOSE OF CONDITIONS
TE_REGISTER
P_REGISTER
CONTINUED_CONDITIONS
EXECUTING STACK_SEGMENT
CURRENT_CONDITION
5.2.3 MANAGE TRAP INTERRUPTS
3-97

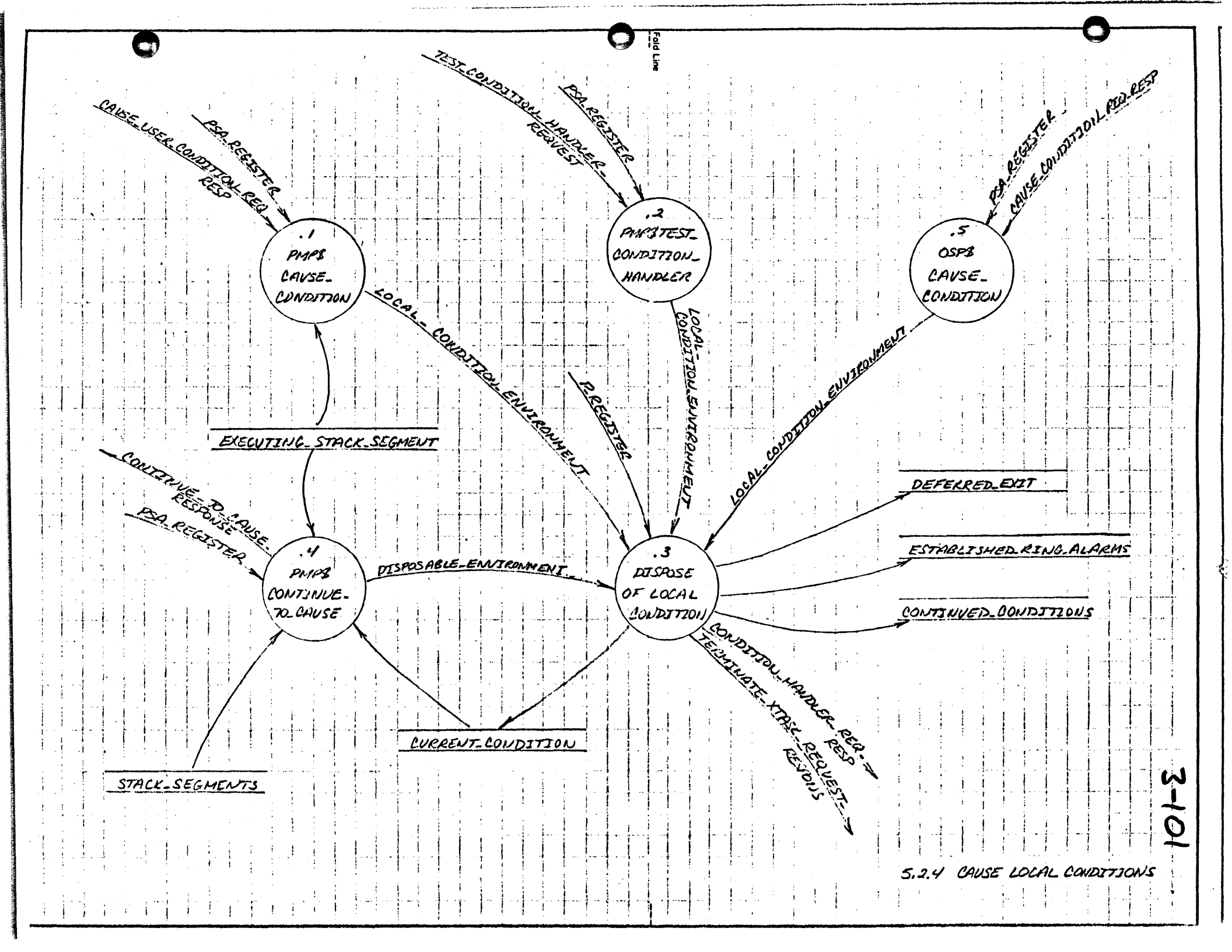

Fold Line
DEFERRED_EXIT_ENVIRONMENT
SIGNAL_ENVIRONMENT
CONTINUED_ENVIRONMENT
DEBUG_INDEX_REGISTER
CONDITION_ENVIRONMENT
.1 DISPOSE OF FREE_FLAG
.3 DISPOSE OF DEBUG TRAP
.5 DISPOSE OF KEYPOINT
TO BE DEFINED
PSA_REGISTER
DEBUG_ENVIRONMENT
FREE_FLAG_ENVIRONMENT
KEYPOINT ENVIRONMENT
RING_ALARMS
.7 ROUTE TRAPS
OTHER_TRAP_ENVIRONMENT
P_REGISTER
CFF_ENVIRONMENT
PIT_ENVIRONMENT
.2 DISPOSE OF CRITICAL FRAME FLAG
.6 DISPOSE OF OTHER TRAPS
ESTABLISHED_RING_ALARMS
CONDITION_ENVIRONMENT
.4 DISPOSE OF PIT TRAP
TRAPPED_STATE
PIT_REGISTER
CONDITION_ENVIRONMENT
ACCUMLATED_TASK_TIME
3-98
5.2.3.1 DISPOSE OF TRAPS



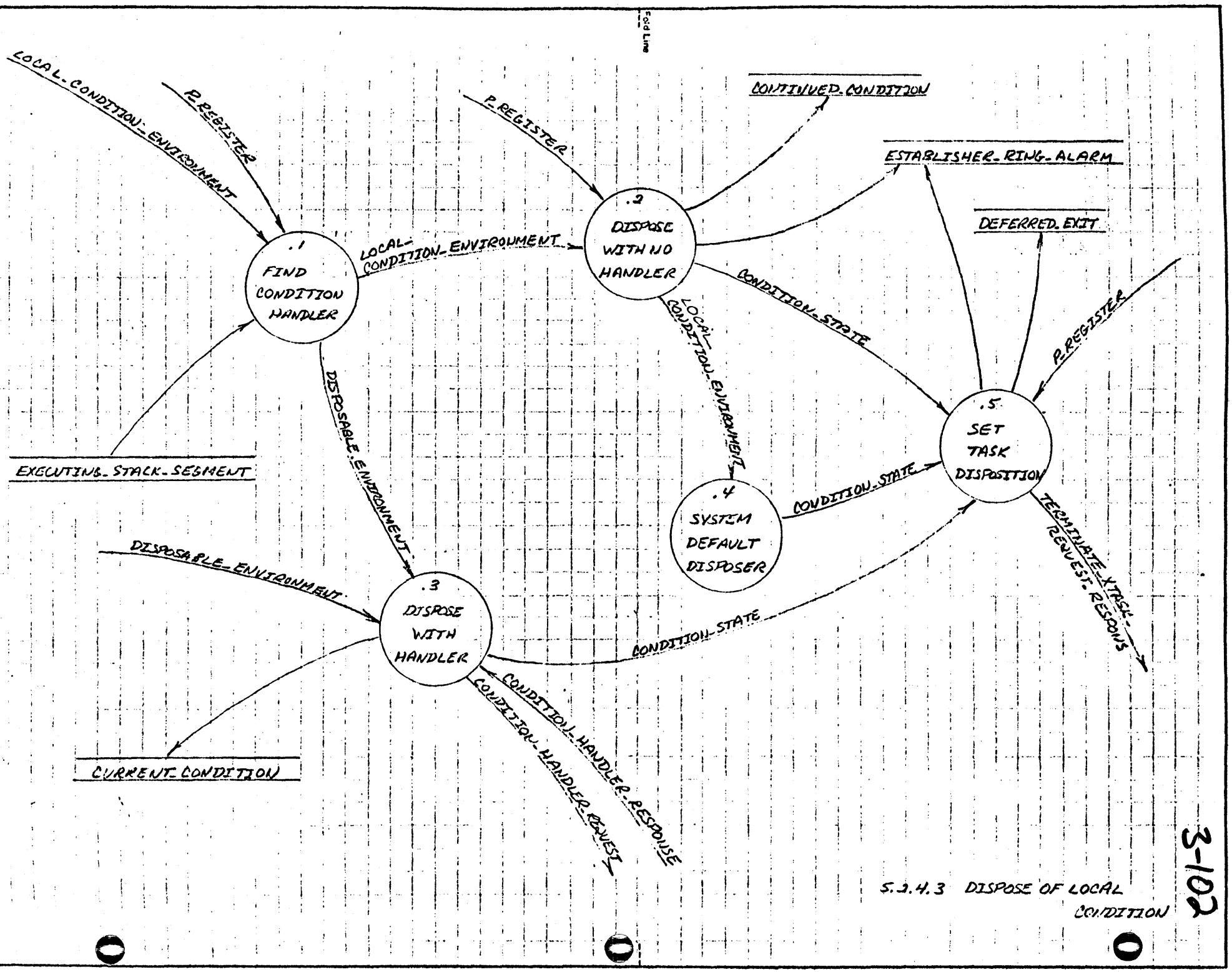
5.2.3.4 DISPOSE OF SIGNALS

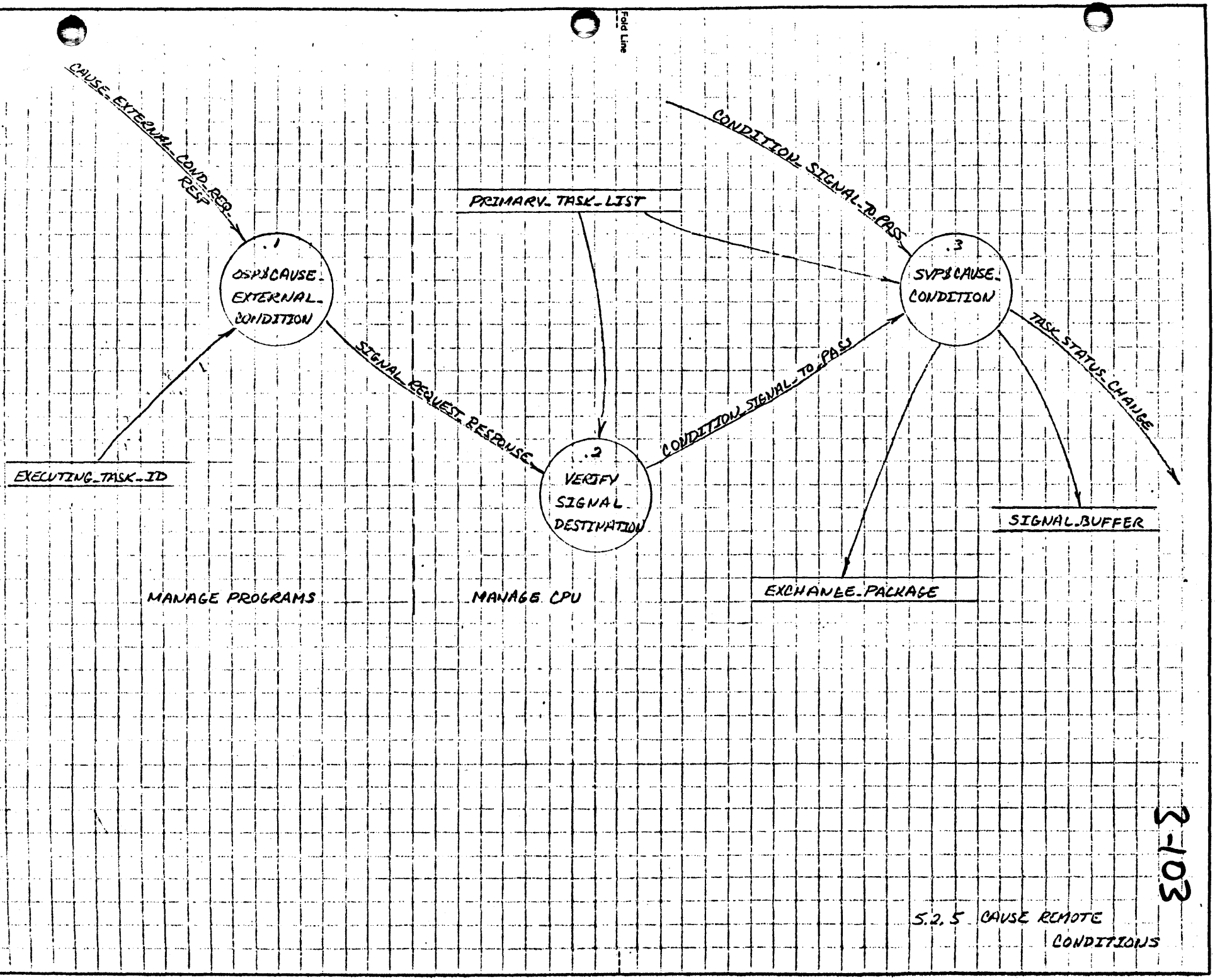


5.2.3.6 DISPOSE OF CONDITION



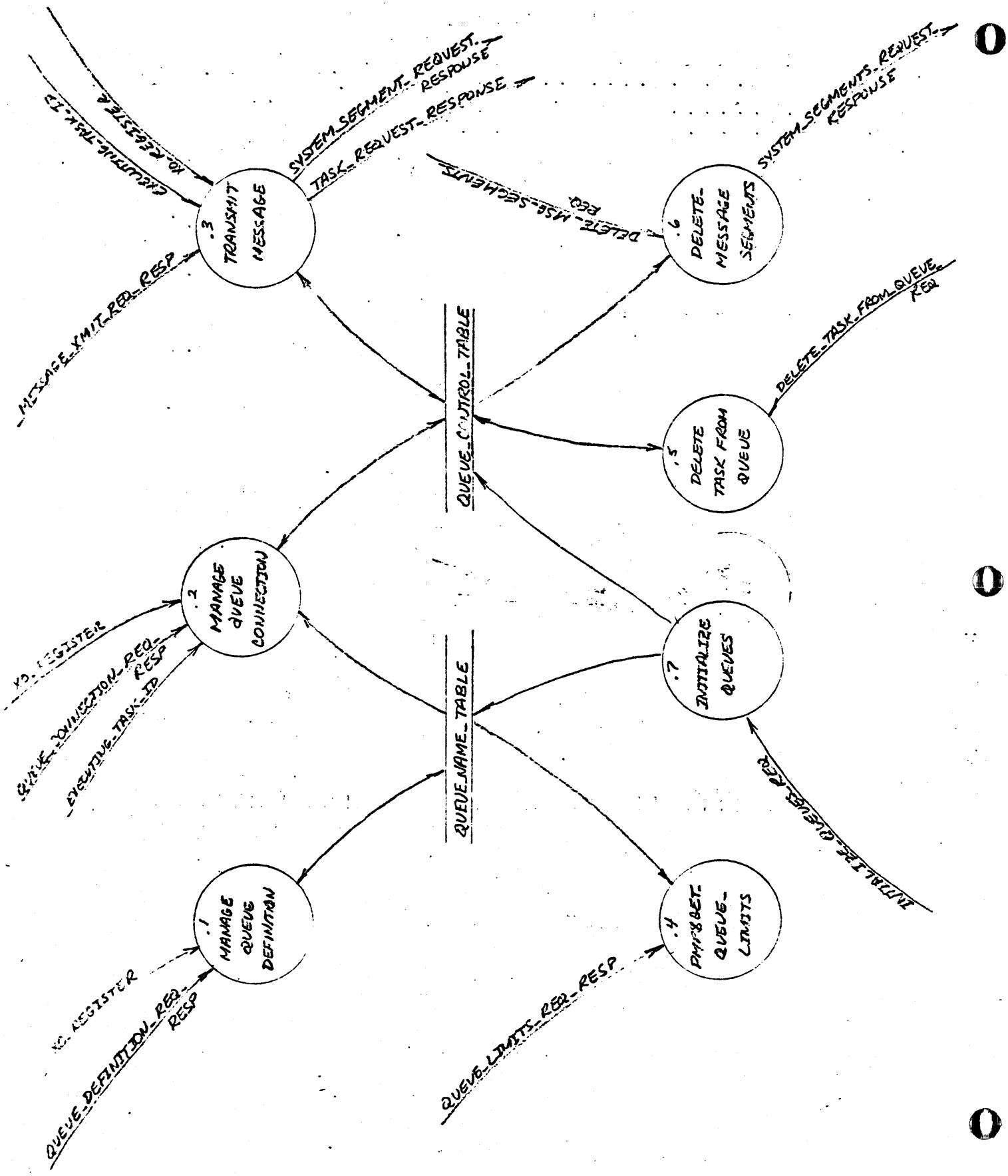
5.2.4 CAUSE LOCAL CONDITIONS

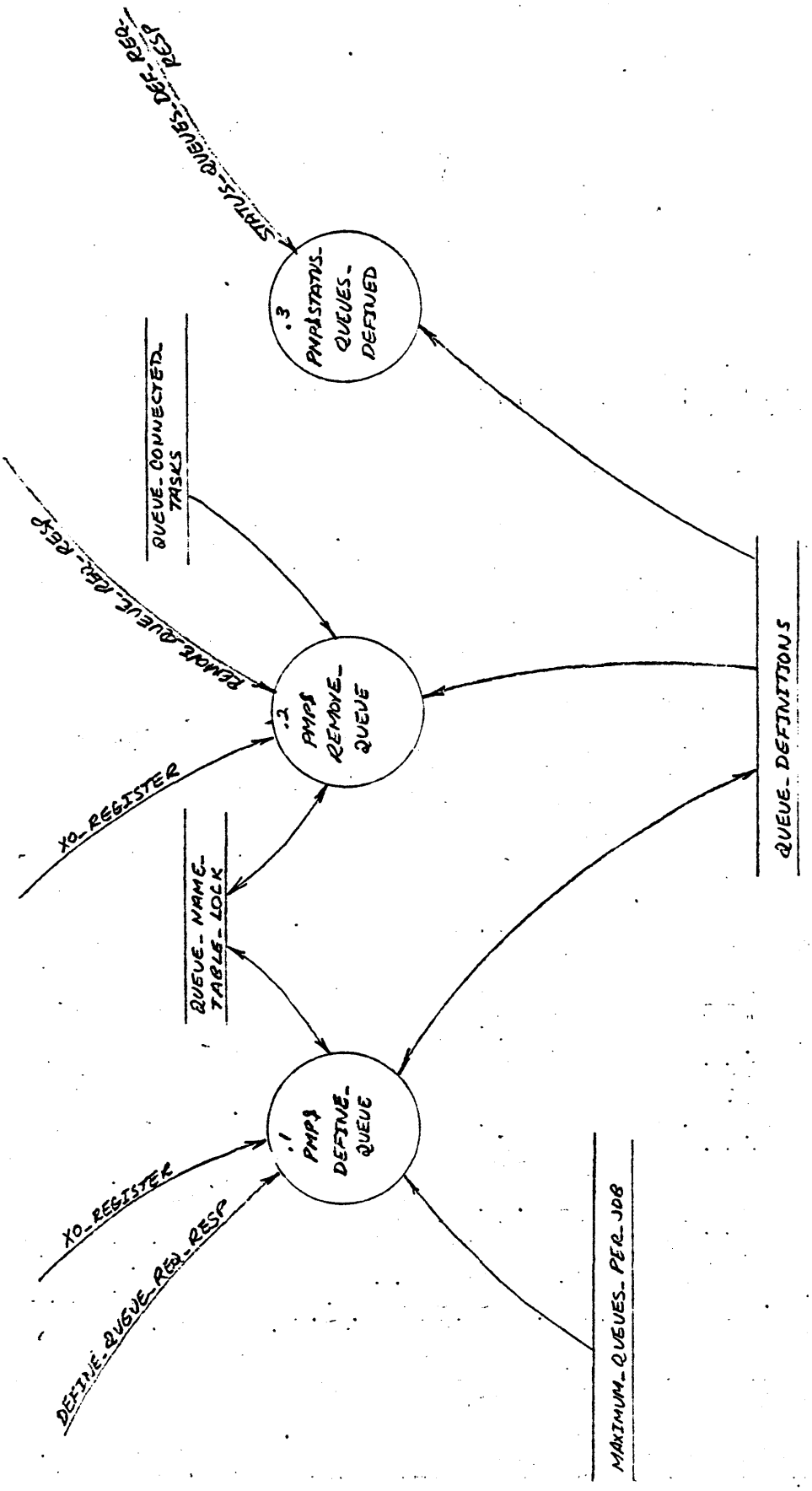

Fold Line
LOCAL_CONDITION_ENVIRONMENT
P_REGISTER
.1
FIND CONDITION HANDLER
EXECUTING_STACK_SEGMENT
LOCAL_CONDITION_ENVIRONMENT
DISPOSABLE_ENVIRONMENT
.2
DISPOSE WITH NO HANDLER
P_REGISTER
CONTINUED_CONDITION
ESTABLISHER_RING_ALARM
DEFERRED_EXIT
CONDITION_STATE
LOCAL_CONDITION_ENVIRONMENT
.4
SYSTEM DEFAULT DISPOSER
CONDITION_STATE
.5
SET TASK DISPOSITION
P_REGISTER
TERMINATE_XTASK_REQUEST_REASONS
DISPOSABLE_ENVIRONMENT
.3
DISPOSE WITH HANDLER
CONDITION_STATE
CONDITION_HANDLER_RESPONSE
CONDITION_HANDLER_REQUEST
CURRENT_CONDITION
5.1.4.3 DISPOSE OF LOCAL CONDITION
3-102

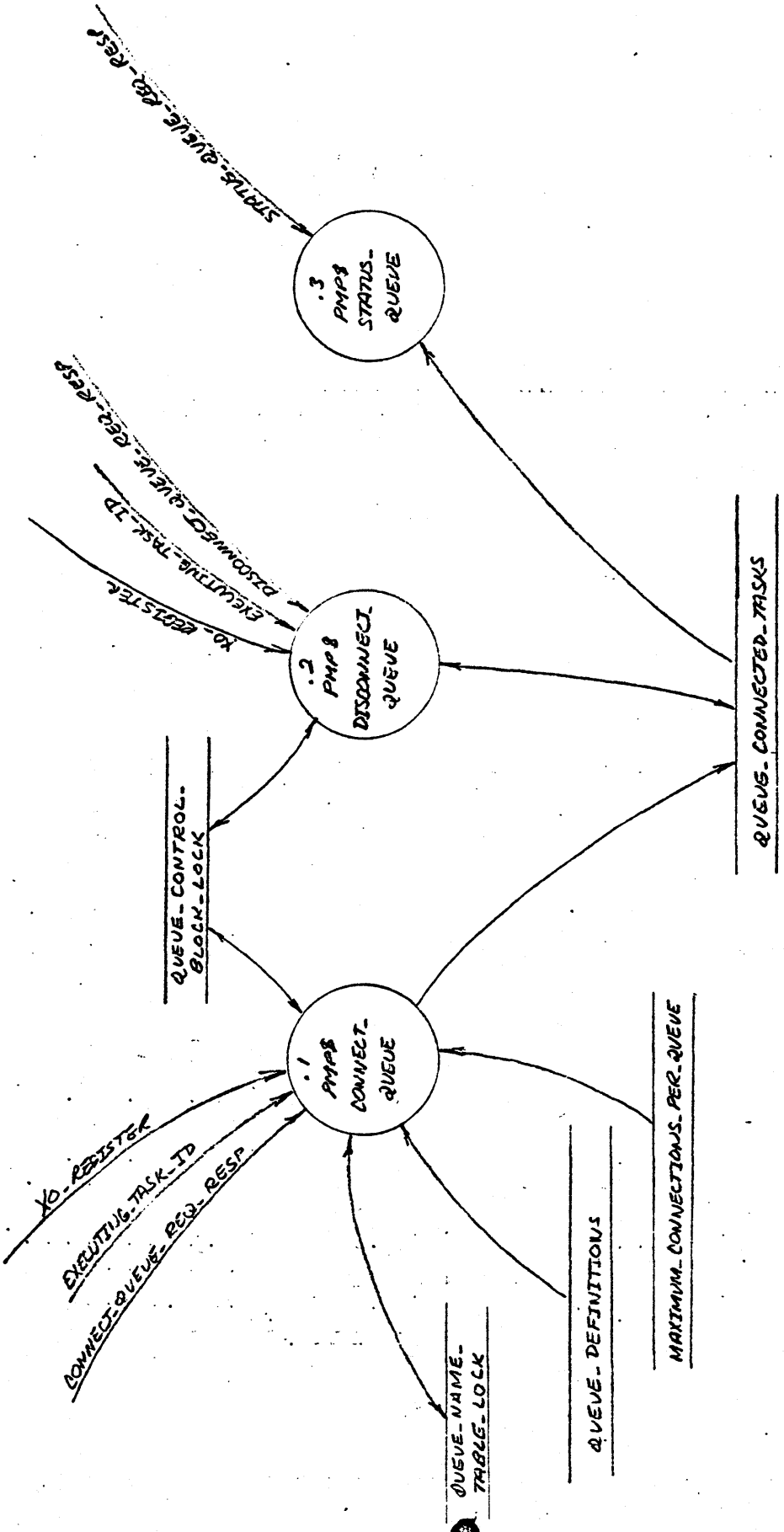


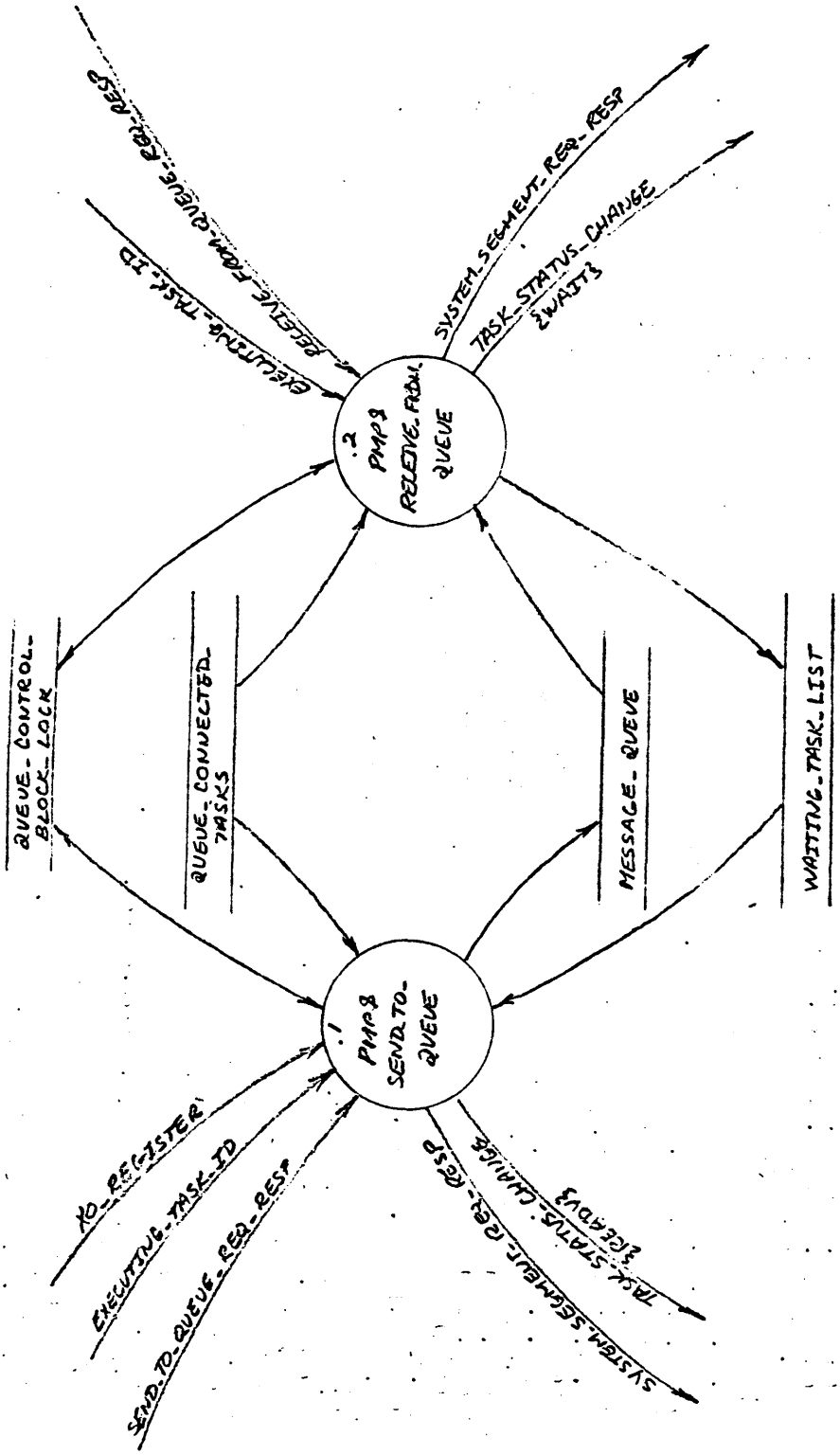
5.2.5 CAUSE REMOTE CONDITIONS

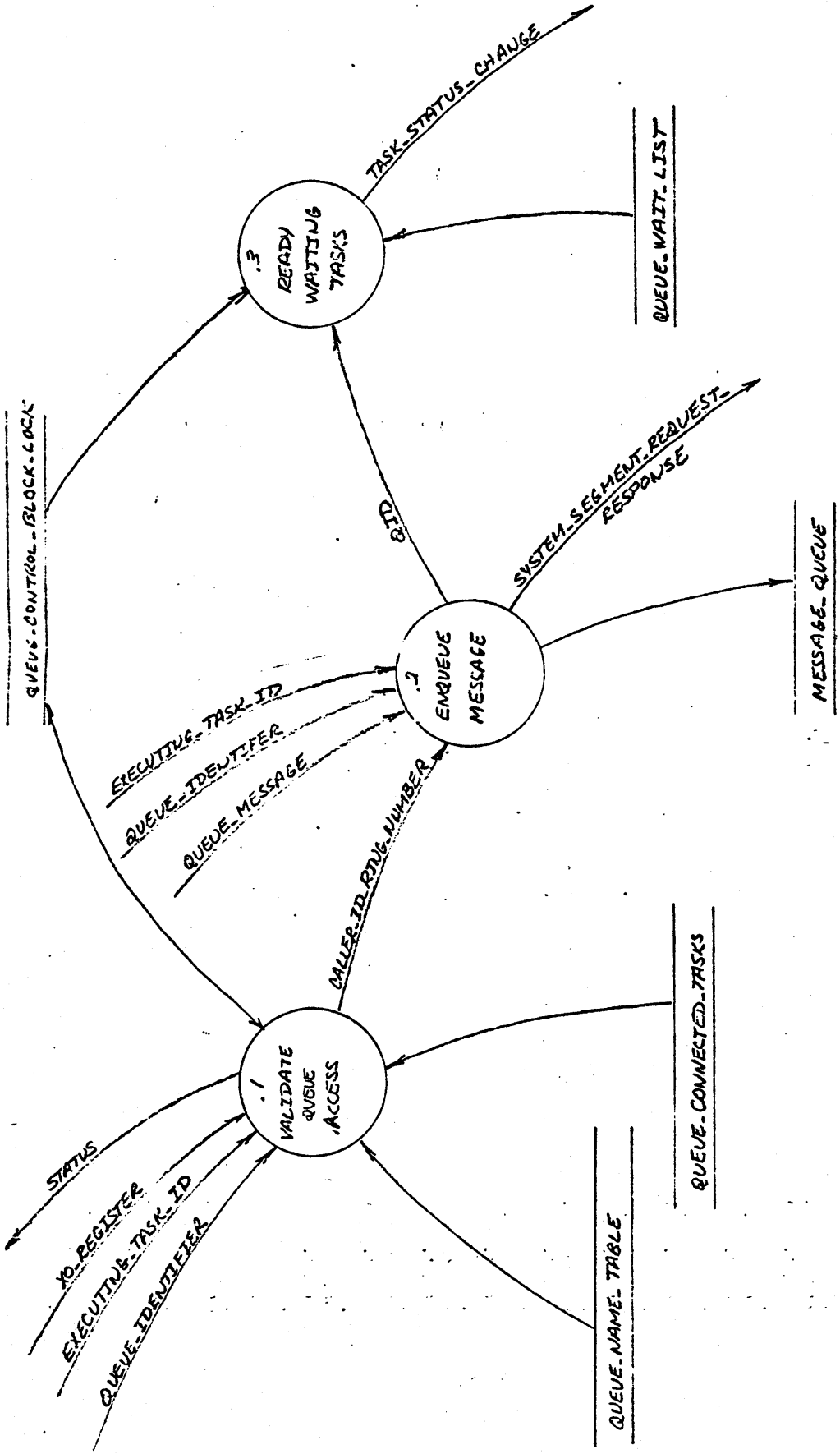
5.3 MANAGE LOCAL MSG QUEUES 12/19/78 L.E. LESKINEN

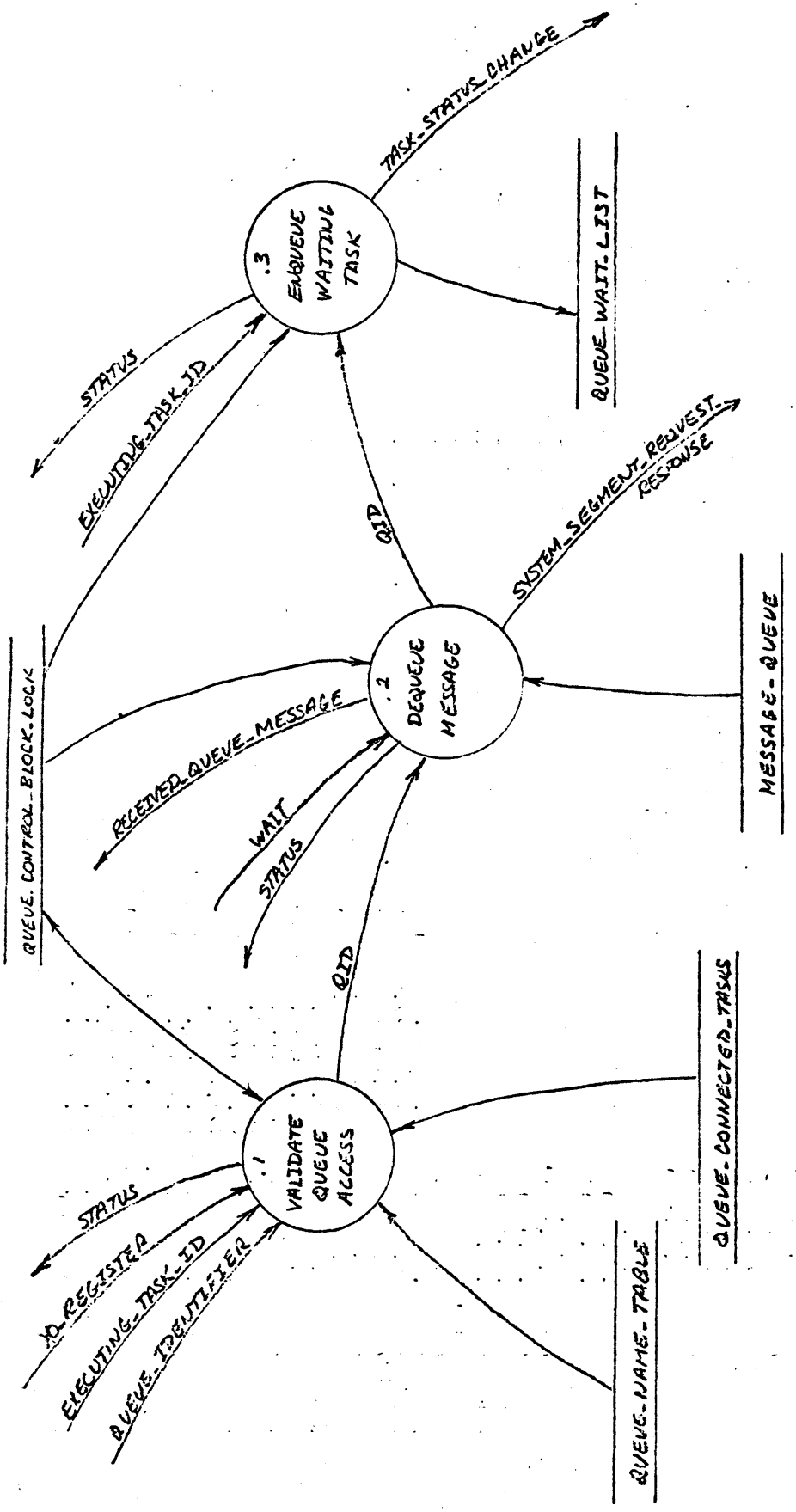



STATUS_QUEUES_DEF_RESP
.3 PNPSTATUS_QUEUES_DEFINED
QUEUE_CONNECTED_TASKS
REMOVE_QUEUE_REQ_RESP
.2 PNPS REMOVE_QUEUE
XQ_REGISTER
QUEUE_NAME_TABLE_LOCK
QUEUE_DEFINITIONS
.1 PNPS DEFINE_QUEUE
XQ_REGISTER
DEFINE_QUEUE_REQ_RESP
MAXIMUM_QUEUES_PER_JOB


STATUS_QUEUE_REQ_RESP
.3 PMP8 STATUS QUEUE
XQ_REGISTER
EXECUTING_TASK_ID
DISCONNECTED_QUEUE_REQ_RESP
.2 PMP8 DISCONNECT QUEUE
QUEUE_CONTROL_BLOCK_LOCK
QUEUS_CONNECTED_TASKS
.1 PMP8 CONNECT QUEUE
CONNECT_QUEUE_REQ_RESP
QUEUE_NAME_TABLE_LOCK
QUEUE_DEFINITIONS
MAXIMUM_CONNECTIONS_PER_QUEUE


.2 PMPS RELEIVE_FROM_QUEUE
.1 PMPS SEND_TO_QUEUE
QUEUE_CONTROL_BLOCK_LOCK
QUEUE_CONNECTED_TASKS
MESSAGE_QUEUE
WAITING_TASK_LIST
EXECUTIVE_TASK_ID
RECEIVE_FROM_QUEUE_REQUEST
SYSTEM_SEGMENT_REQ_RESP
TASK_STATUS_CHANGE_SWAITS
NO_REGISTER
EXECUTIVE_TASK_ID
SEND_TO_QUEUE_REQ_RESP
SYSTEM_SEGMENT_REQ_RESP
TASK_STATUS_CHANGE_SWAITS


.3
READY WAITING TASKS
TASK_STATUS_CHANGE
QUEUE_WAIT_LIST
QUEUE_CONTROL_BLOCK_LOCK
QID
.2
ENQUEUE MESSAGE
SYSTEM_SEGMENT_REQUEST_RESPONSE
MESSAGE_QUEUE
EXECUTING_TASK_ID
QUEUE_IDENTIFIER
QUEUE_MESSAGE
CALLER_ID_RNVS_NUMBER
.1
VALIDATE QUEUE ACCESS
QUEUE_CONNECTED_TASKS
STATUS
NO_REGISTER
EXECUTING_TASK_ID
QUEUE_IDENTIFIER
QUEUE_NAME_TABLE


.3 ENQUEUE WAITING TASK
TASK_STATUS_CHANGE
QUEUE_WAIT_LIST
STATUS
EXECUTING_TASK_ID
QID
SYSTEM_SEGMENT_REQUEST_RESPONSE
.2 DEQUEUE MESSAGE
MESSAGE_QUEUE
QUEUE_CONTROL_BLOCK_LOCK
RECEIVED_QUEUE_MESSAGE
WAIT STATUS
QID
.1 VALIDATE QUEUE ACCESS
QUEUE_CONNECTED_TASKS
STATUS
XD_REGISTER
EXECUTING_TASK_ID
QUEUE_IDENTIFIER
QUEUE_NAME_TABLE

```
accumulated_task_time : FILE =
  time;
accumulated_task_time : ELEMENT =
  time;
activities : FILE =
  number_of_activities +
  1<activity>number_of_activities;
activity : FILE =
  (*time_wait + time !
  *task_termination + callee_local_task_id !
  *file_activity + file_identifier !
  *local_queue_message + queue_identifier);
address_type : ELEMENT =
  (procedure_address !
  data_address);
await_activity_comp_request : FLOW =
  activities;
await_activity_comp_response : FLOW =
  completed_activity_index;
await_activity_request : FLOW =
  activity;
await_task_termination_req : FLOW =
  local_task_id;
callees_local_task_id : FLOW =
  local_task_id;
callees_local_task_id : FLOW =
  local_task_id;
callee_exchange_package : FILE =
  exchange_package;
callee_list : FILE =
  callee_list;
callee_list : FILE =
  <local_task_id>;
callee_local_task_id : FILE =
  local_task_id;
callee_ordinal : ELEMENT =
  local_task_id;
callee_parameters : FLOW =
  task_call_parameters +
  program_descriptor +
  callers_task_status_pointer;
callee_pst_ordinal : FLOW =
  local_task_id;
callee_task_id_resp : FLOW =
  local_task_id;
callee_task_status : FLOW =
  task_status;
callee_xp_pointer : FLOW =
  ^exchange_package;
callend_environment : FLOW =
  callend_signal +
  trapped_sfsa;
callend_request : FLOW =
  callee_task_id;
callend_response : FLOW =
  status;
callend_signal : FLOW =
  callend_signal_id +
```

```
callend_signal_body;
(clend_signal_body : FLOW =
  callee_task_id +
  caller_task_id;
callers_task_status_pointer : ELEMENT =
  ^task_status;
caller_dispatchability : FLCW =
  global_task_id(caller's) +
  (ready !
  wait));
caller_id_ring_number : ELEMENT =
  ring_number;
caller_local_task_id : ELEMENT =
  local_task_id;
caller_pst_ordinal : FLOW =
  local_task_id;
call_ordinal : FLOW =
  local_task_id;
call_relationship : FILE =
  *local_task_id +
  caller_local_task_id +
  callee_list;

call_relationships : FILE =
  2<call_relationship>maximum_tasks_per_job;
cause_condition_request : FLCW =
  (job_resource_condition !
  access_method_condition);
cause_condition_response : FLOW =
  status;
cause_external_condition_req : FLOW =
  condition +
  destination_task_id;
cause_external_condition_resp : FLOW =
  status;
cause_local_cond_req_resp : FLCW =
  (cause_user_condition_req_resp !
  test_condition_handler_request !
  cause_condition_req_resp !
  continue_to_cause_req_resp);
cause_remote_condition_req_resp : FLOW =
  (cause_external_cond_req_resp !
  pass_condition_to_task_req);
cause_user_condition_request : FLOW =
  user_defined_condition +
  condition_info;
cause_user_condition_response : FLOW =
  status;
cff_environment : FLOW =
  trapped_sfsa;
cl_services_request : FLOW =
  (SEE: NOS/180 ERS - Program Interface)
  (scan_parameter_list_req_resp !
  test_parameter_req_resp !
  test_name_req_resp !
  get_set_count_req_resp !
  get_value_count_req_resp !
  get_value_req_resp);
```

```
completed_activity : ELEMENT =
  1 .. number_of_activities;
condition : FLOW =
  (all_conditions !
  primary_conditions !
  secondary_conditions !
  job_resource_condition !
  access_method_condition !
  segment_access_condition !
  user_defined_condition);
condition_environment : FLOW =
  trapped_sfsa +
  condition_sfsa +
  condition +
  condition_info;
condition_handler : ELEMENT =
  pointer_to_procedure;
condition_handler_active : FILE =
  primary_conditions +
  secondary_conditions;
condition_handler_list : FILE =
  <established_condition_handler>;
condition_handler_request : FLOW =
  condition_sfsa +
  condition +
  condition_info;
condition_handler_response : FLOW =
  status;
condition_info : ELEMENT =
  pointer_to_data;
condition_management_req_resp : FLOW =
  (estab_cond_handler_req_resp !
  disestab_cond_handler_req_resp !
  cause_local_cond_req_resp !
  cause_remote_condition_req_resp);
condition_sfsa : ELEMENT =
  ^stack_frame_save_area;
condition_signal : FLOW =
  condition_signal_id +
  condition;
condition_signal_environment : FLOW =
  trapped_sfsa +
  condition_signal;
condition_signal_to_pass : FLOW =
  condition +
  source_task_id +
  destination_task_id;
condition_state : FLOW =
  condition_sfsa +
  status;
condition_to_test : FLOW =
  (primary_conditions !
  secondary_conditions !
  job_resource_condition !
  access_method_condition !
  segment_access_condition);
cond_handler_environment : FLOW =
  condition_environment +
```

```
  established_descriptor +
  established_descriptor_sfsa;
connect_queue_request : FLCW =
  queue_name;
connect_queue_response : FLCW =
  queue_identifier +
  status;
continued_condition : FILE =
  ((debug + continued_debug) !
  job_resource_condition !
  access_method_condition);
continued_conditions : FILE =
  <continued_condition>;
continued_debug : FILE =
  debug_sfsa +
  debug_index;
continued_environment : FLCW =
  trapped_sfsa;
continue_to_cause_request : ELEMENT =
  control_flow;
continue_to_cause_response : FLCW =
  status;
current_condition : FILE =
  handler_active +
  condition_sfsa +
  condition +
  condition_info +
  established_descriptor_sfsa +
  established_descriptor{starting point for continue to cause}
    {established_descriptor search};
debug_environment : FLCW =
  trapped_sfsa;
debug_index : ELEMENT =
  0 .. 31;
debug_sfsa : FILE =
  stack_frame_save_area;
deferred_exit : FILE =
  status +
  terminate_requestor +
  ring_number;
deferred_exit_environment : FLCW =
  trapped_sfsa;
defined_entry_point_table : FILE =
  to_be_defined;
define_queue_request : FLCW =
  queue_name +
  removal_bracket +
  usage_bracket;
define_queue_response : FLCW =
  status;
delete_msg_segments_req : ELEMENT =
  control_flow;
delete_task_from_queue_req : FLCW =
  task_id;
destination_task_id : FLOW =
  task_id;
disconnect_queue_request : FLCW =
  queue_identifier;
```

```
disconnect_queue_response : FLOW =
  status;
disestab_condition_handler_req : FLOW =
  condition;
disestab_condition_handler_resp : FLOW =
  status;
dispatchable_callee : FLOW =
  global_task_id +
  execution_priority;
dispatchable_callee : FLOW =
  global_task_id +
  execution_priority;
disposable_environment : FLOW =
  local_condition_environment +
  established_descriptor +
  established_descriptor_sfsa;
established_condition_handler : FILE =
  established_boolean +
  established_descriptor_stack +
  condition_handler +
  condition_handler_active +
  condition;
established_descriptor : ELEMENT =
  ^established_condition_handler;
established_descriptor_sfsa : ELEMENT =
  ^stack_frame_save_area;
established_descriptor_stack : ELEMENT =
  ^established_condition_handler;
established_ring_alarm : FILE =
  estab_ring_alarm_boolean +
  critical_frame_flag;
established_ring_alarms : FILE =
  2<established_ring_alarm>max_user_ring;
establish_condition_handler_req : FLOW =
  condition +
  condition_handler +
  established_descriptor;
establish_condition_handler_rsp : FLOW =
  status;
exchange_package_ptr : FLOW =
  pointer_to_exechange_package;
execute_request : FLOW =
  callee_parameters +
  execution_relationship;
execute_request_status : FLOW =
  status;
execute_response : FLOW =
  execute_request_status +
  callee_task_status +
  callee_task_id_resp;
executing_local_task_id : FILE =
  local_task_id;
executing_ring_number : ELEMENT =
  ring_number;
execution_ordinal : FLOW =
  local_task_id;
execution_relationship : ELEMENT =
  (wait :
```

```
  nowait);
ting_task : FLOW =
  global_task_id +
  local_task_id;
exiting_task_id : FLOW =
  local_task_id;
exit_request : FLOW =
  status;
first_established_descriptor : ELEMENT =
  established_descriptor;
frame_descriptor : FILE =
  critical_frame_flag +
  on_condition_flag +
  x_starting +
  x_terminating +
  x_terminating);
free_flag_environment : FLOW =
  trapped_sfsa;
general_wait_list : FILE =
  activities;
get_program_response : FLOW =
  program_descriptor;
global_task_ids : FILE =
  <global_task_id>;
head_condition_handler_list : FILE =
  ^established_condition_handler;
initialize_queues_req : FLOW =
(control_flow !
  maximum_queues_per_job);
keypoint_environment : FLOW =
  trapped_sfsa;
loaded_address : ELEMENT =
  (pointer_to_procedure !
  pointer_to_data);
load_request : FLOW =
  xdcled_name +
  address_type;
load_request_response : FLOW =
  load_request +
  load_response;
load_response : FLOW =
  loaded_address +
  status;
local_condition_environment : FLOW =
  condition +
  condition_info +
  condition_sfsa;
local_queue_message : ELEMENT =
  queue_identifier;
local_task_id : ELEMENT =
  0 .. maximum_tasks_per_job;
manage_task_environ_req_resp : FLOW =
  (execute_request_response !
exit_request !
  load_request_response);
manage_task_execution_req_resp : FLOW =
  (terminate_callee_req !
  await_task_termination_req_resp !
```

```
  get_program_req_resp !
  await_activity_comp_req_resp);
maximum_connections_per_queue : ELEMENT =
  20;
maximum_queued_items : ELEMENT {message_blocks or waiting tasks}
  =
  100;
maximum_segments_per_message : ELEMENT =
  20;
max_ring : ELEMENT = 15;
max_user_ring : ELEMENT = 13;
message_block : FILE =
  sender_id +
  sender_ring +
  segment_offset +
  message_type +
  (constant_message !
  pointer_message !
  passed_segments !
  shared_segments);
message_queue : FILE =
  0<message_block>maximum_items_per_queue;
message_queue_request_response : FLOW =
  (queue_definition_req_resp !
  queue_connection_req_resp !
  message_xmit_rec_resp !
  queue_limits_req_resp !
  delete_msg_segments_req_resp);
message_type : ELEMENT =
  (no_message !
  constant !
  pointer !
  shared_segment !
  passed_segment);
message_xmit_req_resp : FLOW =
  (send_to_queue_req_resp !
  receive_from_queue_req_resp);
min_ring : ELEMENT = 1;
non_dispatchable_task : FLOW =
    (wait)
  global_task_id +
  (time);
number_connected_tasks : ELEMENT =
  0 .. maximum_connections_per_queue;
number_defined_queues : ELEMENT =
  0 .. maximum_queues_per_job;
number_of_activities : ELEMENT =
  1 .. n;
number_queued_messages : ELEMENT =
  0 .. maximum_queued_items;
number_waiting_tasks : ELEMENT =
  0 .. maximum_queued_items;
other_secondary_condition : FLOW =
  (privileged_inst_fault !
  unimplemented_instruction !
  inter_ring_pop !
  divide_fault !
  arithmetic_overflow !
```

```
  exponent_overflow !
  exponent_underflow !
  f_p_loss_signif !
  f_p_indefinite !
  arithmetic_loss_signif !
  invalid_bdp_data);
other_secondary_conditions : FLCW =
  [privileged_inst_fault] +
  [unimplemented_instruction] +
  [inter_ring_pop] +
  [divide_fault] +
  [arithmetic_overflow] +
  [exponent_overflow] +
  [exponent_underflow] +
  [f_p_loss_signif] +
  [f_p_indefinite] +
  [arithmetic_loss_signif] +
  [invalid_bdp_data];
other_trap_environment : FLCW =
  other_secondary_condition +
  trapped_sfsa;
passed_message_segment : ELEMENT =
  pointer_to_cell;
passed_message_segments : FLCW =
  number_of_segments +
  0<passed_message_segment>maximum_segments_per_message;
passed_segments : FILE =
  number_of_segments +
  0<segment_descriptor>maximum_segments_per_message;
pit_environment : FLOW =
  trapped_sfsa;
primary_conditions : FLOW =
  monitor_condition_reg_image;
privilege_level : FLOW =
  min_ring .. max_ring;
process_registers : FILE =
  {SEE: MIGDS}
  (p_register{used to derive executing ring number} +
  psa_register {previous save area pointer - A2} +
  x0_register +
  te_register{trap enables} +
  tef_register{trap enable flip-flop} +
  pit_register {process interval timer} +
  debug_index_register);
program_descriptor : FILE =
  defined_in_ERS;
program_services_req_resp : FLCW =
  {SEE: NOS/180 ERS - Program Interface}
  {get_time_req_resp !
  get_date_req_resp !
  get_microsecond_clock_req_resp !
  get_os_version_req_resp);
pst_ordinals : FLOW =
  caller_pst_ordinal +
  callee_pst_ordinal;
queued_items : FILE =
  message_queue +
  queue_wait_list;
```

```
queue_connected_task : FILE =
  local_task_id;
queue_connected_tasks : FILE =
  0<queue_connected_task>maximum_connections_per_queue;
queue_connection_req_resp : FLOW =
  (connect_queue_req_resp !
  disconnect_queue_req_resp !
  status_queue_req_resp);
queue_control_block : FILE =
  *queue_identifier +
  queue_connected_tasks +
  queued_items;
queue_control_table : FILE =
  0<queue_control_block>maximum_queues_per_job;
queue_definition : FILE =
  *queue_identifier +
  defined_boolean +
  queue_name +
  removal_bracket +
  usage_bracket;
queue_definitions : FILE =
  0<queue_definition>maximum_queues_per_job;
queue_definition_req_resp : FLOW =
  (define_queue_req_resp !
  remove_queue_req_resp !
  status_queues_def_req_resp);
queue_identifier : ELEMENT =
  0 .. maximum_queues_per_job;
queue_limits : FLOW =
  maximum_queues_per_job +
  maximum_connections_per_queue +
  maximum_queued_items (per queue);
queue_limits_request : ELEMENT = control_flow;
queue_limits_req_resp : FLOW =
  queue_limits_request +
  queue_limits_response;
queue_limits_response : FLOW =
  queue_limits;
queue_message : FLOW =
  segment_offset +
  message_type +
  (constant_message !
  pointer_message !
  passed_message_segments !
  shared_message_segments);
queue_name_table : FILE =
  maximum_queues_per_job(job initiator or job template) +
  maximum_connections_per_queue(job initiator or job template) +
  maximum_queued_items(job initiator or job template) +
  queue_definitions;
queue_wait_list : FILE =
  0<waiting_task>maximum_items_per_queue;
received_queue_message : FILE =
  sender_id +
  sender_ring +
  segment_offset +
  message_type +
  (constant_message !
```

```
  pointer_message |
  passed_message_segments |
  shared_message_segments:
receive_from_queue_request : FLOW =
  queue_identifier +
  (wait | nowait);
receive_from_queue_response : FLOW =
  received_queue_message +
  status;
removal_bracket : ELEMENT =
  ring_number;
remove_queue_request : FLOW =
  queue_name;
remove_queue_req_resp : FLOW =
  remove_queue_request +
  remove_queue_response;
remove_queue_response : FLOW =
  status;
requestee_task_id : FLOW =
  task_id;
requestor_task_id : FLOW =
  task_id;
returned_callee : FLOW =
  local_task_id;
returning_callee : FLOW =
  local_task_id;
ring_alarm : FILE =
  ring_alarm_boolean +
  free_flag;
ring_alarms : FILE =
  2<ring_alarm>max_user_ring;
ring_number : ELEMENT = min_ring .. max_ring;
secondary_conditions : FLOW =
  {user_condition_register_image}
  [free_flag] +
  [critical_frame_flag] +
  [process_interval_time] +
  [debug] +
  [keypoint] +
  [other_secondary_conditions];
sender_id : FILE =
  local_task_id;
sender_ring : ELEMENT =
  ring_number;
send_to_queue_request : FLOW =
  queue_identifier +
  queue_message;
send_to_queue_response : FLOW =
  status;
sfsa_pointer : ELEMENT =
  ^stack_frame_save_area;
shared_message_segment : ELEMENT =
  pointer_to_cell;
shared_message_segments : FLOW =
  number_of_segments +
  0<shared_message_segment>maximum_segments_per_message;
shared_segments : FILE =
  number_of_segments +
```

```
  0<segment_descriptor>maximum_segments_per_message;
signal : FLOW =
  signal_identifier +
  signal_body;
signal_body : FLOW =
  (callend_signal_body !
  terminate_signal_body !
  condition_signal_body !
  [other_signal_body]);
signal_buffer : FILE =
  <signal>;
signal_buffer_pointer : ELEMENT =
  ^signal_buffer;
signal_environment : FLOW =
  trapped_sfsa;
signal_identifier : ELEMENT =
  [callend_signal_id !
  terminate_signal_id !
  condition_signal_id !
  (other)];
stack_frame : FILE =
  head_condition_handler_list +
  [(condition_handler_list) +
  (stack_frame_save_area) +
  (automatic_variables)];
stack_frames : FILE =
  <stack_frame>;
stack_frame_save_area : FILE =
  p_register_image +
  vmid +
  csp_register_image(dynamic space pointer - A0) +
  csf_register_image(current space pointer - A1) +
  psa_register_image(previous save area pointer - A2) +
  frame_descriptor +
  user_mask +
  user_condition_register_image +
  monitor_condition_reg_image +
  3<a_register_image>a_terminating +
  0<x_register_image>x_terminating;
status_queues_def_response : FLOW =
  number_defined_queues;
status_queue_request : FLOW =
  queue_identifier;
status_queue_response : FLOW =
  status +
  [number_connected_tasks +
  number_queued_messages +
  number_waiting_tasks];
task_control_environment : FILE =
  task_parameter_blocks +
  call_relationships +
  loader_tables;
task_control_request_response : FLOW =
  (manage_task_environ_req_resp !
  manage_task_execution_req_resp);
task_local_data : FILE =
  accumulated_task_time +
  <task_local_static_data> +
```

```
  task_local_heap_data;
sk_local_heap_data : FILE =
 {the heap for dynamically allocated task local data for task monitor and}
  {task services}
  ;
task_local_static_data : FILE =
 {compile-time initialized static data of task monitor and task services}
  ;
task_parameter_block : FILE =
  *local_task_id +
 parameter_string +
  program_descriptor +
 task_status +
  callers_task_status_pointer +
 task_state;
task_parameter_blocks : FILE =
 2<task_parameter_block>maximum_tasks_per_job;
task_param_block_ordinal : ELEMENT =
 local_task_id;
task_state : ELEMENT =
 (unestablished !
  established !
 called !
  executing !
 terminating_normally !
  terminating_abnormally);
task_status : FLOW =
 (complete !
 incomplete) +
  status;
task_to_delete : FLOW =
  global_task_id;
terminated_task_id : FLOW =
  local_task_id;
terminate_callee_req : FLOW =
  local_task_id;
terminate_requestor_task_id : FLOW =
  task_id;
terminate_signal : FLOW =
  terminate_signal_id +
 terminate_signal_body;
terminate_signal_body : FLOW =
 requestor_task_id +
  requestee_task_id;
terminate_xtask_request : FLOW =
  terminate_requestor_task_id +
 status;
terminate_xtask_response : FLOW =
 (continue{task_is_terminating} !
  establish_deferred_exit);
terminating_task : FLOW =
  global_task_id +
 local_task_id;
rminating_task_id : FLOW =
 local_task_id;
test_condition_handler_request : FLOW =
 condition_to_test +
  condition_info +
```

```
  test_sfsa_pointer:
test_sfsa_pointer : ELEMENT =
  ^stack_frame_save_area{dummy};
te_register : FILE =
  tef_register +
  ted_register{trap enable delay};
time : ELEMENT =
  milliseconds;
tpb_ordinal : FLOW =
  local_task_id;
trapped_sfsa : ELEMENT =
  ^stack_frame_save_area;
trapped_state : FLOW =
  trapped_sfsa +
  status;
usage_bracket : ELEMENT =
  ring_number;
waiting_task : FILE =
  local_task_id +
  global_task_id;
wait_for_callee : FILE =
  local_task_id;
xp_ordinal : FLOW =
  local_task_id;
```

```
backing_store_id : ELEMENT =
  file_identifier;
dispatchable_task_list : FILE =
  <ready_string>;
exchange_packages : FILE =
  <task_exchange_package>;
executing_local_task_id : FILE =
  local_task_id;
executing_task : FILE =
  executing_global_task_id +
  executing_local_task_id;
execution_priority : ELEMENT =
  lowest_priority .. highest_priority{defined by manage cou};
FILE_TABLES : FILE =
  <file_descriptor_table> +
  <file_device_descriptor_table> +
  <file_allocation_table>;
global_program_descriptor : FILE =
  program_descriptor{defined by manage programs};
global_task_id : FILE =
  *local_task_id +
  pti_ordinal +
  sequence_number;
initial_exchange_package : FILE =
  task_exchange_package{appropriately initialized by system genreation};
initial_execution_priority : ELEMENT =
  execution_priority;
initial_quantum : ELEMENT =
  time;
initial_task_environment : FILE =
  initial_exchange_package +
  <initial_task_local_static> +
  initial_quantum +
  initial_execution_priority;
initial_task_local_static : FILE =
  {a file containing compile-time initialized data - static segment -for}
  {task monitor or task services. NOTE: some mechanism is required to be }
  {able to map the file to a specific segment.}
  ;
JOB_INHERITED_PARAMETERS : FILE =
  job_limits +
  initial_task_environment +
  global_program_descriptor +
  task_services_entry_definitions;
job_limits : FILE =
  maximum_tasks_per_job +
  maximum_queues_per_job +
  maximum_connections_per_queue +
  maximum_items_per_queue +
  other_job_maximums;
local_task_id : ELEMENT =
  0 .. maximum_tasks_per_job;
maximum_queues_per_job : ELEMENT =
  40;
maximum_tasks_per_job : ELEMENT =
  20;
primary_task : FILE =
  *pti_ordinal +
```

```
  established +
  execution_priority +
  exchange_package +
  sequence_number +
  quantum +
  (some_job_identification{possibly job monitor global_task_id} !
  caller_global_task_id);
primary_task_list : FILE =
  <primary_task>;
process_segment_table : FILE =
  *execution_priority +
  *local_task_id +
  <pst_entry>;
process_segment_tables : FILE =
  1<process_segment_table>maximum_tasks_per_job;
PROGRAM_DATA : FILE =
  object_libraries +
  object_files +
  load_map;
pst_entry : FILE =
  *segment_number +
  segment_descriptor +
  soft_segment_attributes +
  backing_store_id;
ready_string : FILE =
  *execution_priority +
  <ptl_ordinal>:
SEGMENT_DEFINITIONS : FILE =
  process_segment_tables;
soft_segment_attributes : FILE =
  (stack !
  job_global_heap !
  task_local_static !
  task_local_heap);
task_call_parameters : FLOW =
  parameter_string;
task_exchange_package : FILE =
  *local_task_id +
  exchange_package;
TASK_LIST : FILE =
  primary_task_list +
  dispatchable_task_list +
  exchange_packages;
```

```
access_attributes : ELEMENT =
   [read] +
   [write] +
   [execute];

address_formulation_records : ELEMENT =
   {for detailed definition see object text format}
   ;

allocated_section_table : FILE =
   <section_allocation>;

allocated_segment_table : FILE =
   <segment_allocation>;
binding_section_address : ELEMENT =
   {binding section address of the module in which the entry point}
   {is defined}
   ost$pva;

currently_available_address : ELEMENT =
   {address of first unallocated byte in segment}
   ost$pva;

defined_entry_point_table : FILE =
   <entry_point_definition>;

entry_external_records : ELEMENT =
   {for detailed definition see object text format}
   (entry_point_record ! external_linkage_chain);

entry_point_address : ELEMENT =
   {virtual address assigned to the entry point}
   ost$pva;

entry_point_address : ELEMENT =
   ost$pva;

entry_point_call_bracket : ELEMENT =
   {rings from which the entry point may be called (and therefore}
   {rings in which the entry point is defined) ... only entry points}
   {defined in the highest ring of the execute bracket have call}
   {brackets}
   ost$ring;

entry_point_cross_reference : FLOW =
   entry_point_name +
   module_name {module in which entry point is defined} +
   entry_point_address +
   <module_name +
   <referenced_address>> {references to entry point};

entry_point_definition : FLOW =
   *entry_point_name +
   entry_point_address +
   binding_section address +    "parameter checking junk"
   *entry_point_ring_number +
   entry_point_call_bracket +
```

```
  global_local_key_lock +
  virtual_machine_kind;

entry_point_dictionary : FLOW =
  <*entry_point_name +
  module_offset>;

entry_point_name : ELEMENT =
  pmt$program_name;

entry_point_ring_number : ELEMENT =
  {ring in which the entry point is defined}
  ost$ring;

external_linkage : ELEMENT =
  {for detailed definition see object text format}
  ;

external_reference_linkage_tabl
 : FILE =
  <external_linkages>;

global_local_key_lock : ELEMENT =
  ost$gl_key;
global_program_description : FLOW =
  load_map_file_name +
  load_map_rewind_option +
  load_map_level +
  preset +
  load_error_action +
  <global_library_names> {global library list};

initial_ring_of_execution : ELEMENT =
  {ring in which new task is to begin executing}
  ost$ring;

library_attributes : FLOW =
  ring_bracket +
  global_local_key_lock +
  execute_privilege;

library_description_table : FILE =
  <*library_name +
  library_attributes +
  module_dictionary +
  entry_point_dictionary>;
loaded_program : FLOW =
  {segmented executable image}
  <segment>;

loaded_segment_allocations : FLOW =
  segment_number +
  segment_attributes +
  segment_length;
load_error_action : ELEMENT =
  (abort_on_warning ! abort_on_error ! abort_on_fatal);

load_map : FILE =
```

```
load_map_statistics +
[module_address] +
[entry_point_address] +
[entry_point_cross_reference];

load_map_file_name : ELEMENT =
  amt$local_file_name;

load_map_level : ELEMENT =
  set of (no_load_map ! statistics ! block_map ! entry_point_map
  ! entry_point_xref);

load_map_rewind_option : ELEMENT =
  (rewind ! no_rewind);

load_map_statistics : FLOW =
  transfer_address +
  loaded_segment_allocations;

maximum_stack_size : ELEMENT =
  ost$segment_length;

module : FLOW =
  (object_module_format ! load_module_format)
  module_preamble +
  text_insertion_records +
  address_formulation_records +
  entry/external_records +
  unprocessed_records +
  transfer_record;

modules : FLOW =
  (object_module ! load_module);
module_address : FLOW =
  module_name +
  <module_section_addresses +
  module_section_attributes>;

module_description : FLOW =
  module_preamble +
  module_format;

module_dictionary : FLOW =
  <*module_name +
  module_offset>;

module_format : ELEMENT =
  (object_module_format ! load_module_format);

module_list_loaded : ELEMENT =
  {sent when module list has been loaded}
  boolean;

module_name : ELEMENT =
  pmt$program_name;
module_offset : ELEMENT =
  ost$segment_offset;
```

```
module_preamble : FLOW =
  module_header +
  <module_section_definition>;

module_section_addresses : ELEMENT =
  {PVA assigned to the section}
  ost$pva;

module_section_attributes : FLOW =
  segment_attributes;
object_file_name : ELEMENT =
  amt$local_file_name;
object_library : FLOW =
  module_dictionary +
  entry_project_dictionary +
  <load_modules>;

object_library_name : ELEMENT =
  amt$local_file_name;

object_list_loaded : ELEMENT =
  {sent when object list has been loaded}
  boolean;

preset : ELEMENT =
  (zero ! indefinite ! infinity);

program_description : FLOW =
  [starting_procedure] +
  [<object_file_name>] {files containing modules to be loaded} +
  [<module_name>] {modules to be loaded from the library list} +
  [<object_library_names>] {local library list} +
  [initial_ring_of_execution] {should this be here?} +
  [task_call_parameters] +
  [maximum_stack_size] +
  [universal_heap_size] +
  [load_map_file_name] +
  [load_map_rewind_option ] +
  [load_map_level] +
  [preset] +
  [load_error_action];
R1 : ELEMENT =
  ost$ring;

R2 : ELEMENT =
  ost$ring;

R3 : ELEMENT =
  ost$ring;

referenced_address : ELEMENT =
  {address at which an external was referenced}
  ost$pva;

rings_loaded : ELEMENT =
  {all rings in which one or more modules have been labeled}
  set_of_ost$ring;
```

```
ring_brackets : FLOW =
  {These ring numbers which define the four ring brackets of a segment ...}
  {Read_bracket  1=<n=<R2}
  {Write_bracket  1=<n=<R1}
  {Execute_bracket  R1=<n=<R2}
  {Call_bracket  R2<n=<R3}
  R1 +
  R2 +
  R3;

section_address : ELEMENT =
  {base address of the section}
  ost$pva;

section_allocation : FLOW =
  section_type +
  section_address +
  section_length +
  access_attributes +
  ring_brackets +
  global_local_key_lock;

section_length : ELEMENT =
  ost$request_length;

section_type : ELEMENT =
  (code_section ! binding_section ! working_storage_section !
  extensible_working_storage_section ! common_block !
  extensible_common_block);

segment : ELEMENT =
  <byte>;

segment_allocation : FLOW =
  access_attributes +
  ring_brackets +
  global_local_key_lock +
  currently_available_address +
  segment_length;

segment_attributes : FLOW =
  access_attributes +
  ring_brackets +
  global_local_key_lock;

segment_length : ELEMENT =
  {number of bytes allocated by the loader}
  ost$segment_length;

segment_number : ELEMENT =
  ost$segment;

starting_procedure : ELEMENT =
  {user supplied name of external procedure at which execution of a}
  {task is to begin}
  pmt$program_name;

task_call_parameters : ELEMENT =
```

```
  {arbitrary byte string}
  ;

text_embedded_library_names : ELEMENT =
  {local file names of libraries that are emitted by compilers in the}
  {loader text}
  amt$local_file_name;

text_insertion_records : ELEMENT =
  {for detailed definitions see object text format}
  (text_records ! replication_records ! bit_insertion_records);

transfer_address : ELEMENT =
  {address at which execution is to begin}
  ost$pva;

transfer_record : ELEMENT =
  {for detailed definition see object text format}
  ;

transfer_symbol : ELEMENT =
  {last entry point name encountered in the transfer records of}
  {all modules loaded}
  pmt$program_name;

universal_heap_size : ELEMENT =
  ost$segment_length;
unprocessed_records : ELEMENT =
  {for detailed definition see object text format}
  (relocation_record ! binding_template ! formal_parameters !
  actual_parameters);
unsatisfied_external_reference : ELEMENT =
  pmt$program_name;

unsatisfied_external_table : FILE =
  <*unsatisfied_external_reference +
  external_reference_linkage_table_index>;

virtual_machine_kind : ELEMENT =
  (CYBER_170 ! CYBER_180);
```

```
PROCEDURE accumulate entry point cross references; { 5.1.1.3.3 }
  FOR all entry point cross references received DO
    IF a chain_for the entry point already exists THEN
      add external linkage cross reference to chain;
    ELSE
      build new entry point cross reference item;
    IFEND;
  FOREND;
  write_load_map (load_map.cross_reference);
PROCEND accumulate entry point cross references;
```

```
PROCEDURE create_stack_segments; { 5.1.1.3.4 }
  FOR each ring within which a_module was loaded DO
    create a stack segment;
    place its addresses in the exchange package;
  FOREND;
PROCEND create_stack_segments;
```

```
PROCEDURE write load map; { 5.1.1.3.5 }
  IF a load map is to be written THEN
    IF statistics are to be written THEN
      write segment allocations on load m write transfer address on load map;
    IFEND;
    IF block map is to be written THEN
      write module_addresses on load map;
    IFEND;
    IF entry points are to be written THEN
      write entry point addresses on load map;
    IFEND;
    IF entry point cross references are to be written THEN
      write entry point cross references on load map;
    IFEND;
    send load map;
  IFEND;
PROCEND write load map;
```

```
PROCEDURE establish transfer address; { 5.1.1.3.6 }
  IF program_description.starting_procedure specified THEN
    transfer symbol := program_description.starting_procedure;
  IFEND;
  IF transfer_symbol defined THEN
    transfer_to_user_code (transfer_address);
    RETURN;
  IFEND;

{ transfer symbol has not already been loaded: search the library list
{ for_it. }

/SEARCH_FOR TRANSFER SYMBOL/
  FOR all libraries in the library description table DO
    search entry point dictionary for_transfer symbol;
    IF transfer_symbol in current library THEN
      load_a_module (module_containg transfer symbol, initial_ring);
      EXIT /SEARCH_FOR TRANSFER SYMBOL/;
    IFEND;
  FOREND /SEARCH_FOR_TRANSFER SYMBOL/;
PROCEND establish transfer address;
```

```
PROCEDURE transfer_to_user_program; { 5.1.1.3.7 }

  establish dummy stack frame in users ring;
  set_up task call parameters as arguments to user task;
  perform outward call to transfer address;
PROCEND transfer_to_user_program;
```

```
PROCEDURE determine entry point address; { 5.1.1.3.8 }
  IF entry point isnt in defined entry point table THEN
    REPEAT
      search library entry point dictionary for_entry point defined in callers
        ring;
    UNTIL library list exhausted OR entry point defined;
  IFEND;
  CASE address_type OF
  =pva=
    RETURN address of_entry point;
  =procedure_=
    build procedure_description of entry point;
    RETURN address_of procedure_description;
  CASEND;
PROCEND determine entry point address;
```

```
PROCEDURE determine_initial_ring_number; {5.1.1.3.1.1 }

  IF (first_object_file.r3 >= callers_ring) AND (callers_ring >
     first_object_file.r2) THEN
     initial_ring := first_object_file.r2;
   ELSE
     IF (first_object_file.r2 >= callers_ring) AND (callers_ring >=
       first_object_file.r1) THEN
       initial_ring := callers_ring;
     ELSE
       initial_ring := first_object_file.r1;
     IFEND;
   IFEND;
PROCEND determine_initial_ring_number;



PROCEDURE load_object_list; { 5.1.1.3.1.2 }
   FOR every_object_file DO
     get_file_attributes (object_file);
     IF initial_ring > current_object_file.r2 THEN
       r := current_object_file.r2;
     ELSE
       r := initial_ring;
     IFEND;
     FOR every_module_on_the file DO
       load_a_module (object file module_ description);
     FOREND;
   FOREND;
   build_library_attribute_table (program_description.library_list);
PROCEND load_object_list;



{   Build library description table: }
{    consists of two parts: }
{     1.Local library list }
{     2.Global library list
{   This dichotomy is maintained in order to allow local libraries }
{   to be added to the local library list during loading. }

PROCEDURE build library attribute table; {5.1.1.3.1.3 }
   FOR all program_description.library_list DO
     get_file_attributes;
     ALLOCATE next slot in local library list;
     fetch library dictionary;
     build library list entry;
   FOREND;
   FOR all global_program.library_list DO
     get_file_attributes;
     ALLOCATE next slot in global library list;
     fetch library dictionaries;
     build library list entry;
   FOREND;
PROCEND build library attribute table;
```

```
PROCEDURE load modules from module_list; { 5.1.1.3.1.4 }

{ Load all modules in the module list in the initial ring of execution. }

  FOR all libraries IN the library description table DO
  /search directory for module_/
    FOR all modules IN module_list DO
      IF module_ already loaded THEN
        CYCLE /SEARCH DIRECTORY FOR MODULE_/
      IFEND;
      IF module_.initial_ring IN current library THEN
        load_a_module (module_, initial_ring);
        mark module_ as already loaded;
      IFEND;
    FOREND /SEARCH DIRECTORY FOR MODULE_/;
  FOREND;
  establish_transfer_symbol (program_description.starting_procedure,
    initial_ring_number);
PROCEND load modules from module_list;



PROCEDURE satisfy external references; { 5.1.1.3.1.5 }
  start := first entry in unsatisfied external table;
  this_iterations_finish := current last entry in unsatisfied external table;
/SATISFY EXTERNALS/
  WHILE TRUE DO
    FOR all libraries within the library description table DO
      FOR start TO this iterations finish DO
        search entry_point_dictionary for_ unsatisfied externals;
        IF found THEN
          load_a_module (module_containing unsatisfied external);
        IFEND;
      FOREND;
    FOREND;
    EXIT /SATISFY EXTERNALS/ WHEN no unsatisfied externals remain;
    start := this_iterations_finish + 1;
    this_iterations_finish := current last entry in unsatisfied external table;
  WHILEND /SATISFY EXTERNALS/;
PROCEND satisfy external references;



PROCEDURE load text; { 5.1.1.3.2.2 }
  CASE record_type OF
  =text_record=
    insert text bytes at specified (section_, offset);
  =replication_record=
    repetitively insert text at specified (section_, offset);
  =bit insertion_record=
    insert bit string at specified (section_, offset + bit_offset);
  CASEND;
PROCEND load_text;
```

```
PROCEDURE formulate_addresses; { 5.1.1.3.2.3 }
  FOR all address formulations IN the record_ DO
    CASE address_types OF
    =ova=
      build forty eight bit ova associated with (value_section, value_offset);
      store it within (destination_section, destination_offset);
    =internal_procedure=
      build sixty four bit code base pointer associated with (value_section,
        value_offset);
      store it within (destination_section, destination_offset);
    =one word external_procedure=
      same as internal_procedure except external flag bit is on;
    =external_procedure=
      build sixty four bit code base pointer associated with (value_section,
        value_offset);
      store it at (destination_section, destination_offset);
      store address of current modules binding_section at (destination_section,
        destination_offset + 10);
    CASEND;
  FOREND;
PROCEND formulate addresses;



PROCEDURE read records not requiring processing; { 5.1.1.3.2.5 }
  { read and discard records }
PROCEND read records not requiring processing;



PROCEDURE save_transfer_address; { 5.1.1.3.2.6 }
  save the last transfer symbol encountered in any_module;
PROCEND save_transfer_symbol;



PROCEDURE add_libraries_to_library_list; { 5.1.1.3.2.1.1 }
  FOR all library records DO
    build_library_attribute_table (text embedded library names);
  FOREND;
PROCEND add_libraries_to_library_list;



PROCEDURE process load_module header; { 5.1.1.3.2.1.2 }
  IF a code_section exists_for the_module THEN
    write code section_allocation IN load_module code_section table;
  IFEND;
  write load map (module_name);
  allocate_sections for_a_module (number_of_sections);
PROCEND process load_module header;
```

```
PROCEDURE process object_module header; {5.1.1.3.2.1.3 }
  write_load_map (module_name);
  allocate_sections for_a_module (number_of_sections);
PROCEND process object_module header;
```

```
await_activity_completion : PROCESS 5.1.2.4 =
  {await_activity_comp_request is a list of activities comparable to}
  {  the general_wait_list.}
  completed_activity := 0;
  copy the user's list of activities to the general_wait_list;
  wait_time := largest_integer;
  {get the shortest time to wait}
  FOR activity_index := 1 TO activity_index > number_of_activities DO
    IF activity is time_wait THEN
      IF time < wait_time THEN
        wait_time := time;
      IFEND;
      get current_time {free running clock};
      add current_time to general_wait_list[activity_index].time;
    IFEND;
  FOREND;
  REPEAT
    activity_index := 1;
    REPEAT
      CASE activity OF
      =task_termination=
       IF (callee_local_task_id is not a callee of this task)
         {callee_local_task_id is not in call_relationships }
         {    [executing_local_task_id].callee_list} THEN
         completed_activity := activity_index;
       IFEND;
      =file_activity=
       IF executing task has the file associated with file_identifier open THEN
         IF file_tables associated with file_identifier reveal no outstanding
           request THEN
           completed_activity := activity_index;
         IFEND;
       ELSE
         reject;
       IFEND;
```

cont'd →

Cont'd

```
    =local_queue_message=
     IF queue_identifier is valid and specifies a defined queue
       {i.e., defined_boolean in the queue_name_table entry corresponding}
       {         to queue_identifier is true} THEN
       IF executing task is connected to the queue
         {i.e., executing_global + local_task_id appear in the }
         {         queue_connected_tasks[queue_identifier]} THEN
         IF there is a message on the message_queue[queue_identifier] THEN
           completed_activity := activity_index;
         ELSE
           prevent access to queue_wait_list[queue_identifier] by other tasks;
           put executing task into the queue_wait_list[queue_identifier];
           permit access to the queue_wait_list[queue_identifier];
         IFEND;
       ELSE
         reject;
       IFEND;
     ELSE
       reject;
     IFEND;
    =time_wait=
     get current_time {free running clock};
     IF current_time >= general_wait_list[activity_index].time THEN
       completed_activity := activity_index;
     IFEND;
    CASEND;
  UNTIL (activity_index > number_of_activities) OR
    (completed_activity <> 0);
  IF completed_activity equals 0 THEN
    IF wait_time <> largest_integer THEN
      pass executing_global_task_id + wait_time as nor_dispatchable_task;
    ELSE
      pass executing_global_task_id + some_arbitrary_time as
        non_dispatchable_task;
    IFEND;
  IFEND;
 UNTIL completed_activity <> 0;
 pass completed_activity as await_activity_comp_resp
    {completed_activity_index};
PROCESS_END;
```

```
await_task_termination : PROCESS 5.1.2.3 =
  {await_task_termination_req is a local_task_id}
  {executing_local_task_id is the requesting task's local_task_id}
  {executing_global_task_id is global_task_ids[executing_local_task_id]}
  {NOTE: modification of the callee_list by returning callees must be}
  {      prevented from inspection in the  IF   thru the pass of non_}
  {      dispatchable_task to ensure that the task does not go into   }
  {      WAIT just after the callee returns. Passing time as a part }
  {      of non_dispatchable_task is an alternate solution.          }

  IF local_task_id is call_relationships[executing_local_task_id].callee_list
    THEN
    put local_task_id into wait_for_callee;
    pass executing_global_task_id as non_dispatchable_task;
  ELSE
    clear {zero} wait_for_callee;
  IFEND;
PROCESS_END;
```

```
close_user_files : PROCESS 5.1.1.2.3.2 =
  {exiting_task_id is the exiting task's local_task_id.}
  {executing_global_task_id is the exiting task's global_task_id.}
  {executing_global_task_id is global_task_ids[exiting_task_id]}
  {open user files are identified by local and global_task_id.}
  FOR each open user file DO
    {close the file.}
    execute any user options associated with the file:
    decrement the file usage count by one:
    remove the task identification from the associated file_tables;
  FOREND;
  pass exiting_task_id + executing_global_task_id as exiting_task;
PROCESS_END;
```

```
determine_caller_execution : PROCESS 5.1.1.1.4.4 =
  {global_task_id is the caller's global_task_id.}
  IF caller_dispachablity is wait THEN
    {make the task non_dispatchable}
    pass global_task_id as non_dispatchable_task;
  IFEND;
PROCESS_END;
```

```
determine_task_execution : PROCESS 5.1.1.1.4.5 =
  {executing_global_task_id identifies the task being returned to.}
  executing_global_task_id is global_task_ids[executing_local_task_id]}
  IF returned_callee is wait_for_callee THEN
    clear wait_for_callee{set it to zero};
  ELSE
    IF wait_for_callee is not clear {zero} THEN
      pass executing_global_task_id as non_dispatchable_task;
    IFEND;
  IFEND;
PROCESS_END;
```

```
disconnect_from_queues : PROCESS 5.1.1.2.3.3 =
  {exiting_task is the exiting task's local and global_task_id}
  FOR each defined queue in the queue_name_table DO
    IF exiting_task is connected to the queue {exiting_task is in}
      {queue_connected_tasks} THEN
      prevent access to the asscciated queue_control_table by other
        tasks;
      IF exiting_task is in the queue_wait_list THEN
        remove exiting_task from the queue_wait_list:
      IFEND;
      {disconnect exiting_task from the queue}
      remove exiting_task from queue_connected_tasks;
      permit access to the associated queue_control_table;
    IFEND;
  FOREND;
  pass exiting_task as terminating_task;
PROCESS_END;
```

```
disestablish_call_relationship : PROCESS 5.1.1.2.2 =
  {call_ordinal is the exiting task's local_task_id.}
  {call_relationship[call_ordinal].caller_local_task_id is the caller's}
  {  local_task_id.}

  {update caller's callee_list}
  prevent modification to the caller's callee_list - caller task or
    anyother task returning to the caller task;
  remove the exiting task {call_ordinal} from the caller's callee_list:
  permit modification to the caller's callee_list;
  pass caller_local_task_id;
PROCESS_END;
```

```
disestablish_system_segment : PROCESS 5.1.1.1.5.1 =
  {terminated_task_id is the terminated task's local_task_id.}
  {callee_process_segment_table is process_segment_table[terminated_task_id]}

  {job_global_data segments are identified by soft_segment_attribute}
  FOR each job_global_data segment in callee_process_segment_table DO
    decrement the usage count for job_global_data segment backing
      store file in file_tables{appropriate file table[backing_store_id]};
  FOREND;

  {task monitor/task services code and binding segments are those}
  {  segments whose segment descriptor's R1 values are <= 3 and  }
  {  control field is executable or binding and VL is not invalid.}
  FOR each task_monitor/task_services code and binding segment in
    callee_process_segment_table DO
    decrement the usage count for segment backing store file
      in file_tables{appropriate file table[backing_store_id]};
  FOREND;

  {return task_monitor/task_services stack segments backing store files.}
  {task_monitor/task_services stack segments are identified by R1 values}
  {  <= 3 and VL is not invalic and soft_segment_attribute of stack.}
  FOR each task_monitor/task_services stack segment in
    callee_process_segment DO
    remove the entry corresponding to the segment's backing_store_id
      from file_tables and release the file space;
  FOREND;

  {return task_monitor/task_services data segments backing store files.}
  {task_monitor/task_services data segments are identified by R1 values}
  {  <= 3 and VL is not invalic and soft_segment_attribute of          }
  {task_mon_serv_data.}
  FOR each task_monitor/task_services data segment in
    callee_process_segment DO
    remove the entry corresponding to the segment's backing_store_id
      from file_tables and release the file space;
  FOREND;
PROCESS_END;



disestablish_tpb : PROCESS 5.1.1.1.5.3 =
  {returning_callee is the returning callee's local_task_id}
  {callee_task_parameter_block is task_parameter_block[returning_callee]}
  set task_state to disestablished in callee_task_parameter_block;
  {callee is finally disestablished in its entirity.}
  pass returning_callee as returned_callee;
PROCESS_END:
```

```
disestablish_user_segments : PROCESS 5.1.1.2.3.1 =
  {exiting_task_id is the exiting task's local_task_id}
  {process_segment_table is exiting task's process_segment_table - }
  {  process_segment_table[exiting_task_id].}

  {user segments are those segments whose segment descriptor R1 value }
  {  is > 3 and the VL is not invalid.                                 }

  {disestablish user stack segments - stack segments are identified by}
  {  soft_segment_attribute of stack.                                  }
  FOR each user stack segment DO
    free the file_tables associated with backing_store_id, returning
      the file space;
    set the segment_descriptor invalid;
  FOREND;

  FOR each valid user segment DO
    decrement the usage count associated with backing_store_id;
    IF (usage count is zero) and (the file associated with backing_store_id
      is not a permanent file) THEN
      return the file(i.e., free the file_tables associated with       }
        {backing_strore_id, returning the file space.}:
    IFEND;
  FOREND;
  pass exiting_task_id;
PROCESS_END;
```

```
establish_callee_exchange_pckg : PROCESS 5.1.1.1.3.2 =
  {xp_ordinal is the callee's local_task_id}
  {callee_excharge_package is exchange_packages[xp_ordinal]}
  copy the initial_exchange_package to callee_exchange_package;
  set callee_exchange_package.a2 {previous save area pointer} to nil;
  compute segment_table_address {process_segment_table rma} from
    xp_ordinal;
  put the segment_table_address in callee_exchage_package;
  initialize any callee_exchange_package registers that are not
    pre-initialized by system_generation;
  pass callee_exchange_package^ as callee_xp_pointer;
PROCESS_END;
```

```
establish_callee_parameters : PROCESS 5.1.1.1.2.2 =
  {tpb_ordinal is the callee's local_task_id}
  {callee_task_parameter_block is task_parameter_blocks[tpb_ordinal]}
  {move callee_paramters to callee_task_parameter_block}
  callee_task_parameter_block.program_descriptor :=
    callee_parameters.program_descriptor;
  callee_task_parameter_block.parameter_string :=
    callee_parameters.parameter_string;
  callee_task_parameter_block.callers_task_status_pointer :=
    callee_parameters.callers_task_status_pointer;
  pass tpb_ordinal as callees_local_task_id;
PROCESS_END;



establish_call_relationship : PROCESS 5.1.1.1.2.1 =
  {callee_ordinal is the callee's local_task_id}
  call_relationships[callee_ordinal].caller_local_task_id :=
    executing_local_task_id;
  set call_relationships[callee_ordinal].callee_list empty;
  {add the callee to the caller's(executing task) callee_list}
  prevent modification to executing task's callee_list by a terminating
    callee.
  call_relationships[executing_local_task_id].callee_list[n] :=
    callee_ordinal{callee_local_task_id};
  permit modification of the executing task's callee_list.
  pass callee_ordinal as tpb_ordinal;
PROCESS_END;



establish_job_shared_segments : PROCESS 5.1.1.1.3.1 =
  {execution_ordinal is the callee's local_task_id}
  {callee_pst is process_segment_table[execution_ordinal]}
  {caller_pst is process_segment_table[executing_local_task_id]}
  {all pst_entry copies are caller_pst[segment_number] to}
  {  callee_pst[segment_number].                          }

  {job_global_data segments are identified by soft_segment_attribute}
  FOR each job_global_data segment in caller_pst DO
    copy the pst_entry from caller_pst to the callee_pst;
    increment the usage ccurt for job_global_data segment backing
      store file in file_tables[appropriate file table[backing_store_id]];
  FOREND;

  {task_monitor/task_services code and binding segments are identified: }
  {  segments whose segment descriptor R1 value is <= 3 and the control}
  {  field is executable or binding and VL is not invalid.}
  FOR each task_monitor/task_services code and binding segment caller_pst DO
    copy pst_entry from caller_pst to the callee_pst;
    increment the usage count for segment's backing store file in
      file_tables[appropriate file table[backing_store_id]];
  FOREND;
PROCESS_END;
```

```
establish_stack_segments : PROCESS 5.1.1.1.3.4 =
  {callee_pst_ordinal is the callee's local_task_id}
  {callee_pst is process_segment_table[callee_pst_ordinal]}
  {callee_xp_pointer points to callee_exchange_package}
  FOR each task_monitor and task_services ring in the callee_pst DO
    create a process_segment_table[callee_pst_ordinal] entry
      for a stack segment;
    process_segment_table[callee_pst_ordinal].soft_segment_attributes :=
      stack;
    open a backing_store file for the stack segment;
    put the file_identifier of the backing store file in the created
      callee_pst entry;
    put the pva of the stack segment to the appropriate tos register
      in the callee_exchange_package[byte_offset = 0];
  FOREND;
  get the stack_segment_number corresponding to callee_exchange_package.
    p_register.ring_number;
  put callee_exchange_package.p_register.ring_number, stack_segment_number,
    and byte_offset of 0 {zero} into
   A0 {dynamic space pointer} and A1 {current stack frame pointer}
   of the callee_exchange_package;
  pass callee_xp_pointer as exchange_package_ptr;
PROCESS_END;
```

```
establish_task_local_segments : PROCESS 5.1.1.1.3.3 =
  {callee_pst_ordinal is the callee's local_task_id}
  {callee_pst is process_segment_table[callee_pst_ordinal]}
  {caller_pst_ordinal is the caller's local_task_id}
  {caller_pst is process_segment_table[caller_pst_ordinal]}
  {initialized_task_mon_serv_data is compile-time initialized static}
  {  residing on some file known to the task initiator.}
  {task_mon_serv_data is segment image of that static}

  {this process is not necessary if task_monitor/task_services have ro}
  {  compile-time initialized static.}

  FOR each task_mon_serv_data segment DO
    copy pst_entry from caller_pst to callee_pst;
    open a backing store file for the task_mon_serv_data segment;
    put the file_identifier of the backing store file in the callee pst_entry;
    copy the initialized_task_mon_serv_data to the backing store file for
      task_mon_serv_data;
  FOREND;
  pass callee_pst_ordinal;
PROCESS_END;
```

```
force_callees_to_terminate : PROCESS 5.1.2.2.2 =
    {a trigger {signal}
  will cause this task to terminate.}
  {force all callees in this task's {executing_local_task_id} to terminate}
  FOR each callee in the executing task's callee_list
    {call_relationships[executiong_local_task_id].callee_list} DO
    pass callee_local_task_ic as terminate_callee_req:
  FOREND;
  pass executing_local_task_ic as local_task_id;
PROCESS_END;



force_callee_termination : PROCESS 5.1.2.2.1 =
  {terminate_callee_req is the callee_local_task_id of the callee to}
  {  terminate.}
  IF callee_local_task_id is in the executing task's callee_list
    {call_relationships[executing_local_task_id].callee_list} THEN
    get the callee_globab_task_ic from global_task_ids
      {global_task_ics[callee_local_task_id]}:
    get callee_execution_priority from the primary_task_list
     {primary_task_list[callee_global_task_id.ptl_ordinal].execution_priority};
    {make the callee dispatchable}
    IF the callee is not in the appropriate ready_string in the
      dispatachable_task_list {ready_string[callee_execution_priority]} THEn
      add the callee{ptl_ordinal} to the ready_string[callee_execution_priority];
    IFEND:
    {a trigger {signal} is necessary to cause the callee task to terminate}
  IFEND;
PROCESS_END;



get_program_descriptor : PROCESS 5.1.2.1 =
  program_descriptor := task_parameter_blocks[executing_task_id.local_task_id].
    callee_parameters.program_cescriptor;
  pass program_descriptor;
PROCESS_END;



make_callee_dispatchable : PROCESS 5.1.1.1.4.3 =
  add the callee[dispatchable_callee.global_task_id.ptl_ordinal} to
    the appropriate ready_string in the dispatchable_task_list
    {ready_string[dispatchable_callee.execution_priority]}:
PROCESS_END;
```

```
make_callee_known : PROCESS 5.1.1.1.4.2 =
  {find an unestablished primary_task_list entry}
  WHILE (ptl_ordinal is < maximum number of primary_tasks) AND
    (the callee is unknown) DO
    IF NOT primary_task_list[ptl_ordinal].established THEN
      {make the callee known in the primary_task_list}
      set primary_task_list[ptl_ordinal].established to true;
      put exchange_package_ptr, initial_execution_priority, initial_quantum
        into primary_task_list[ptl_ordinal];
      increment primary_task_list[ptl_ordinal].sequence_number by 1;
      put ptl_ordinal and sequence_number {callee global_task_id} into
        global_task_ids {global_task_ids[callees_local_task_id]};
    ELSE
      increment ptl_ordinal;
    IFEND;
  WHILEND;

  IF the callee is known THEN
    put callees_local_task_id into callee_local_task_id;
    pass ptl_ordinal and execution_priority as dispatchable_callee;
  ELSE
    reject;
  IFEND;
PROCESS_END;
```

```
make_task_unknown : PROCESS 5.1.1.2.4.2 =
  {task_to_delete is global_task_id}

  {disestablish the task's ptl_entry}
  set primary_task_list[task_to_delete.ptl_ordinal].established to false;
  pass task_to_delete.ptl_ordinal as non_dispatchable_task;
PROCESS_END;
```

```
notify_caller_of_termination : PROCESS 5.1.1.2.4.1 =
  {caller_global_task_id is global_task_ids[caller_local_task_id]}
  IF the caller is not in dispatchable_task_list THEN
    {make the caller dispatchable by adding the caller to the }
    {  ready_string[execution_priority] - execution_priority is found in }
    {  primary_task_list[caller_global_task_id.ptl_ordinal]}
    add the caller{caller_global_task_id.ptl_ordinal} to
      the appropriate ready_string in the dispatchable_task_list
      {ready_string[execution_priority]};
  IFEND;
  pass terminating_task.local_task_id as terminating_task_id;
  pass terminating_task.global_task_id as task_to_delete;
PROCESS_END;
```

```
return_task_status : PROCESS 5.1.1.1.5.2 =
  {returning_callee is the returning callee's local_task_id}
  return/pass callee_task_id;
  {i.e., move task_parameter_blocks[returning_callee].callee_task_status}
  {  to task_parameter_blocks[returning_callee].caller_task_status_pointer}
  {  setting complete to true.}
  pass returning_callee;
PROCESS_END;
```

```
return_to_user_proc_caller : PROCESS 5.1.1.2.1.2 =
  {This process has the effect of returning to the caller (i.e., task}
  {initiator) of first user procedure - executing any block exit     }
  {processing condition handlers that may be outstanding.            }

  sfsa := sfsa_pointer;
  {last stack_frame to unwind has a psa_register_image = nil}
  WHILE {stack_frame is not the last user stack_frame} (sfsa <> nil) DO
    IF {a pop would cross rings}
      (sfsa.ring_number < sfsa^.psa_register_image.ring_number) THEN
      {move the current stack_frame to the next higher ring stack_segment}
        {round destination dynamic space pointer to a word boundary}
      sfsa^.dsp_register_image := (sfsa^.dsp_register_image + 7) mod 8;
      #bytes := dsp_register.offset - csf_register.offset;
       FOR i := 0 TO (#bytes - 1) DO
         sfsa^.dsp_register_image[i] := csf_register^[i];
       FOREND;
      {create a new stack_frame_save_area in that stack_segment by moving  }
      {the executing registers to that area.  this may be best accomplished}
      {by calling a procedure - having the procedure move the stack frame  }
      {save area which is laid down to the new stack_frame_save_area.      }
      new_sfsa := (sfsa^.dsp_register_image + #bytes + 7) mod 8;
      move_executing_registers (new_sfsa);
      {compute new pointers for the registers in the dummy sfsa.}
      new_dsp := (new_sfsa + #size(new stack_frame_save_area));
      new_csf := sfsa^.dsp_register_image;
      new_p_register := #loc(ring_crossing_return) + 2;
      new_p_register.ring_number := sfsa^.psa_register.ring_number;
      new stack_frame_save_area.psa_register_image := sfsa;
      new stack_save_frame_area.p_register_image := new_p_register;
      new stack_frame_save_area.dsp_register_image := new_dsp;
      new stack_frame_save_area.csf_register_image := new_csf;
      psa_register := new_sfsa;
      {set the appropriate tos register to point to the new stack_frame.}
      tos_register[new_csf.ring_number] := new_csf;
      /ring_crossing_return/
      return to the new stack_frame;
    IFEND;
    pop the stack_frame: {pop will cause block exit processing          }
       {condition handlers and continued condition handlers to execute.}
    sfsa := psa_register;
  WHILEND; {alternative: this process could perform the same functions}
      {as condition processing. first determining if a block exit}
      {would occur and then finding the condition handler - calling}
      {that handler directly from this process.}
  {return to the last stack_frame in the task}
PROCESS_END;
```

```
set_execution_relationship : PROCESS 5.1.1.1.4.1 =
  {callees_local_task_id is the callee's local_task_id.}
  IF execution_relationship is wait THEN
    put callees_local_task_id in wait_for_callee;
  ELSE
    put zero in wait_for_callee;
  IFEND;
  pass callees_local_task_id;

  {executing_global_task_id is global_task_ids[executing_local_task_id]}
  pass execution_relationship + executing_global_task_id as
    caller_dispatchability;
PROCESS_END;
```

```
terminate_task : PROCESS 5.1.2.2.3 =
  {local_task_id is this task's local_task_id}
  {task_parameter_block[local_task_id] is this task's task_parameter_block}
  IF task_parameter_block[local_task_id].task_state is NOT
    (terminating_normally OR terminating_abnormally) THEN
    pass status{abnormal - forced termination} as exit_request;
  IFEND;
PROCESS_END;
```

```
update_task_state : PROCESS 5.1.1.2.1.1 =
  {executing_local_task_id is the exiting task's local_task_id}
  {task_state and status are in task_parameter_block[executing_local_task_id]}
  prevent access to exiting task's task_state by another execution instance
    of update_task_state;
  IF task_state is not (terminating_normally OR terminating_abnormally) THEN
    IF exit_request[status] is normal THEN
      set task_parameter_block[executing_local_task_id].task_state
        to terminating_normally;
    ELSE
      set task_parameter_block[executing_local_task_id].task_state
        to terminating_abnormally;
    IFEND;
    permit access to exiting task's task_state;
    set exiting task's task_status to exit_request[status]
      setting complete to true;
    sfsa_pointer := psa_register;
    pass sfsa_pointer;
    pass executing_local_task_id as exiting_task_id;
  IFEND;
  permit access to exiting task's task_state;
PROCESS_END;
```

```
validate_program_execution : PROCESS 5.1.1.1.1 =
  {local_task_id becomes callee's local_task_id}
  prevent access to task_parameter_blocks by other instances of
    execution attempting to establish a task - the executing task or
    another task in the job.
  local_task_id := 0;
   {search for an unestablished task_parameter_block}
  REPEAT
    local_task_id := local_task_id + 1;
  UNTIL (task_parameter_blocks[local_task_id].task_state =
    unestablished) or (local_task_id = maximum_tasks_per_job);
  IF task_parameter_blocks[local_task_id].task_state =
    unestablished THEN
    {set callee_task_parameter_block established}
    task_parameter_blocks[local_task_id].task_state = established;
    permit access to task_parameter_blocks;
    put local_task_id in callee_local_task_id;
    pass execute_request_status{normal};
    pass local_task_id as callee_task_id_resp;
    pass callee_task_status{incomplete};
    pass local_task_id as callee_ordinal;
    pass local_task_id as execution_ordinal;
  ELSE
    permit access to task_parameter_blocks;
    pass execute_request_status{abnormal - max tasks exceeded};
  IFEND;
PROCESS_END;


wait_for_callees : PROCESS 5.1.1.2.1.3 =
  {exiting_task_id is the exiting task's local_task_id}
  {executing_global_task_id is global_task_ids[exiting_task_id]}

  {NOTE: modification of the callee_list by returning callees must be}
  {      prevented from inspection in the WHILE thru the pass of non_}
  {      dispatchabe_task to ersure that the task does not go into   }
  {      WAIT when there are no callees.  Passing time as a part of  }
  {      non_dispatchable_task is an alternate solution.             }
  WHILE exiting task's callee_list
    {call_relationship[exiting_task_id].callee_list} is not empty DO
    pass executing_global_task_id as non_dispatchable_task:
    {returning callees will make this task dispatchable}
  WHILEND;
  pass exiting_task_id as call_ordinal;
  pass exiting_task_id + executing_global_task_id as exiting_task;
PROCESS_END;



wait_for_io_complete : PROCESS 5.1.1.2.1.4 =
  {exiting_task - local and global_task_id will identify files for}
  {the exiting task in file_tables}
  WHILE there is i/o active on any file DO
    pass exexiting_task.global_task_id as non_dispatchable_task;
    {i/o completion will make this task dispatchable}
  WHILEND;
  pass exiting_task.local_task_id as exiting_task_id;
PROCESS_END;
```

```
delete_task_from_queue : PROCESS 5.3.5 =
  FOR qid := 0 TO queue_name_table.maximum_queues_per_job DO
    IF queue_name_table.queue_definition[qid].defined_boolean = true THEN
      search queue_control_table[qid].queue_connected_tasks for task_id
        UNTIL (task_id is found connected to a queue) OR
          (all queue_connected_tasks are searched);
      IF task_id is found in queue_connected_tasks THEN
        set queue_control_table[qid].queue_control_block_lock;
        search queue_control_table[qid].queue_wait_list for task_id
          UNTIL (task_id is found in the queue_wait_list) OR
            (the entire queue_wait_list is searched);
        IF task_id is found in the queue_wait_list THEN
          remove the task from the queue_wait_list;
        IFEND;
        remove the task from queue_connected_task(i.e., disconnect the
          task from the queue);
        clear queue_control_table[qid].queue_control_block_lock;
      IFEND;
    IFEND;
  FOREND;
PROCESS_END:1
dequeue_message : PROCESS 5.3.3.2.2 =
  WHILE (received_queue_message has not been passed) AND
    (status is normal) DO
    IF (queue_control_table.queue_control_block[qid].message_queue
      contains a message_block) THEN
      dequeue the message_block from the message_queue to a local
        message_block;
      clear queue_control_block[qid].queue_control_block_lock;
      received_queue_message.sender_id := message_block.sender_id;
      received_queue_message.sender_ring := message_block.sender_ring;
      received_queue_message.segment_offset := message_block.segment_offset;
      received_queue_message.message_type := message_block.message_type;
      CASE message_block OF
      =constant_message=
       received_queue_message.constant_message := message_block.constant_message;
      =pointer_message=
       received_queue_message.pointer_message := message_block.pointer_message;
         message_block.pointer_message;
      =passed_segment=
       {have manage_sements remove the passed segment(s) from the job ard}
       {add the passed segment(s) to the caller's process segment table}
       {(address space).}
       received_queue_message.passed_message_segments.number_of_segments :=
         message_block.passed_segments.number_of_segments;
       FOR i := 0 TO message_block.passed_segments.number_of_segments DO
         issue system_segment_request(message_block.passed_segments.
           passed_segment[i]):
         received_queue_message.passed_message_segments.
           passed_message_segment[i] := system_segment_response;
         {NOTE: manage_segments may return abnormal status as part of }
         {system_segment_response - in which case the FOR loop process}
         {must be reversed to remove passed segments from the task ard}
         {add back into the job.  The message_block must be placed    }
         {back on queue(first out).  The abnormal status must then be }
         {passed to the caller (receive_from_queue_resp).}
       FOREND;
```

Cont'd →

Cont'd

```
      =shared_segment=
       {have manage_sements remove the shared segment(s) from the job ard}
       {add the shared segment(s) to the caller's process segment table}
       {(address space).}
       received_queue_message.shared_message_segments.number_of_segments :=
         message_block.shared_segments.number_of_segments;
       FOR i := 0 TO message_block.shared_segments.number_of_segments DO
         issue system_segment_request(message_block.shared_segments.
           shared_segment[i]};
         received_queue_message.shared_message_segments.
           shared_message_segment[i] := system_segment_response;
         {NOTE: manage_segments may return abnormal status as part of }
         {system_segment_response - in which case the FOR loop process}
         {must be reversed to remove shared segments from the task and}
         {add back into the job.  The message_block must be placed    }
         {back on queue(first out).  The abnormal status must then be }
         {passed to the caller (receive_from_queue_resp).}
       FOREND;
      CASEND;
      pass received_queue_message;
      pass status(normal);
    ELSE
      IF no_wait THEN
      {form a no_message response}
      received_queue_message.message_type := no_message;
      received_queue_message.sender_id := executing_task_id;
      received_queue_message.sender_ring := caller_id_ring_number;
      clear queue_control_block(qid).queue_control_block_lock;
      pass received_queue_message;
        pass status(normal);
      ELSE
        pass qid;
      IFEND;
    IFEND;
  WHILEND;
PROCESS_END;1
dispose_of_condition_signal : PROCESS 5.2.3.4.3 =
  transform condition_signal to a condition + condition_info;
  condition_sfsa := trapped_sfsa(condition_signal_environ);
  condition_environment := condition + condition_info + condition_sfsa +
    trapped_sfsa;
  pass condition_environment;
PROCESS_END;1
```

Cont'd →

```
dispose_of_continued_conditions : PROCESS 5.2.3.5 =
  localize continued_conditions; {dispose_of_conditions may be placing}
  {conditions into the global continued_conditions as this process is }
  {disposing of conditions.}
  REPEAT
    IF condition is debug THEN
      condition_sfsa := ^continued_condition.continued_debug.debug_sfsa;
      condition_info := ^continued_condition.continued_debug.debug_index;
    ELSE
      condition_sfsa := nil;
      condition_info := nil;
    IFEND;
    pass condition_environment{condition + condition_info + condition_sfsa +
      continued_environment-trapped_sfsa};
  UNTIL all continued_conditions have been disposed of;
PROCESS_END;1
dispose_of_critical_frame_flag : PROCESS 5.2.3.1.2 =
  trapped_sfsa is cff_environment;
  executing_ring_number := p_register.ring_number;
  IF (trapped_sfsa^.psa_register_image.ring_number > executing_ring_number) AND
    (established_ring_alarms[executing_ring_number] = true) THEN
    {set a ring_alarm and clear the established ring alarm}
    set the free_flag in the first stack_frame_save_area of the next
      highest stack_segment;
    ring_alarms[executing_ring_number + 1] := true;
    established_ring_alarms[executing_ring_number] := false;
  IFEND;
```

Cont'd →

```
  IF on_condition_flag is set in trapped_sfsa^.frame_descriptor THEN
    condition := block_exit_processing;
    condition_info := nil;
  condition_sfsa := trapped_sfsa;
    pass condition_environment{condition + condition_info + condition_sfsa +
      trapped_sfsa};
  IFEND;
PROCESS_END;1
dispose_of_debug_trap : PROCESS 5.2.3.1.3 =
  {transform debug trap environment to debug condition_environment}
  debug_index := debug_index_register;
  condition := debug;
  condition_info := #loc(debug_index);
  condition_sfsa := trapped_sfsa{debug_environment};
  pass condition_environment{condition + condition_info + condition_sfsa +
    trapped_sfsa};
PROCESS_END;1
dispose_of_deferred_exit : PROCESS 5.2.3.3 =
  executing_ring_number := p_register.ring_number;
  IF executing_ring_number > deferred_exit.ring_number THEN
    issue terminate_xtask_request (deferred_exit.terminate_requestor,
      deferred_exit.status);
    IF terminate_xtask_response = establish_deferred_exit THEN
      deferred_exit.ring_number := executing_ring_number;
      {establish a ring alarm}
      set established_ring_alarms[executing_ring_number] := true;
      {find the last stack_frame_save_area in this stack_segment}
      sfsa := trapped_sfsa{deferred_exit_environment};
      WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
        sfsa := sfsa^.psa_register_image;
      WHILEND;
      {set the critical_frame_flag in the last sfsa of the stack_segment}
      sfsa^.frame_descriptor.critical_frame_flag;
    ELSE
      deferred_exit.ring_number := 0;
    IFEND;
  IFEND;
  trap_state := normal status;
  pass trap_state;
PROCESS_END;1
dispose_of_free_flag : PROCESS 5.2.3.1.1 =
  {dispose of signals}
  signal_environment := trapped_sfsa{free flag environment};
  pass signal_environment;
  IF ring_alarms[executing_ring_number] = true then
    ring_alarms[executing_ring_number] := false;
    {dispose of deferred exit and continued conditions}
    deferred_exit_environment := trapped_sfsa{free flag environment};
    pass deferred_exit_environment;
    continued_environment := trapped_sfsa{free flag environment};
    pass continued_environment;
  IFEND;
PROCESS_END;1
```

Cont'd →

Cont'd

```
dispose_of_other_traps : PROCESS 5.2.3.1.6 =
  condition_info := nil;
  condition := other_secondary_condition;
  condition_sfsa := trapped_sfsa{other_trap_environment};
  pass condition_environment{condition + condition_info + condition_sfsa +
    trapped_sfsa};
PROCESS_END;1
dispose_of_pit_trap : PROCESS 5.2.3.1.4 =
  {process interval timer interrupt - pit}
  accumulated_task_time := accumulated_task_time + initial_pit_value:
  (initial_pit_value - pit_register) to pit_register;
  trap_state := normal status;
  pass trap_state;
  pass trap_disposition;
PROCESS_END;1
dispose_of_terminate : PROCESS 5.2.3.4.2 =
  get {terminate}requestor_task_id from terminate_signal;
  set status to abnormal{being terminated by another task in the job};
  enable trap interrupts{te_register.tef};
  issue terminate_xtask_request (requestor_task_id, status):
  disable trap interrupts{te_register.tef};
  IF terminate_xtask_response = establish_deferred_exit THEN
    IF executing_ring_number > deferred_exit.ring_number THEN
      {establish a ring alarm for deferred exit}
      deferred_exit.status := status;
      deferred_exit.terminate_requestor := requestor_task_id;
      deferred_exit.ring_number := executing_ring_number;
    {establish a ring alarm}
    set established_ring_alarms[executing_ring_number] := true;
    {find the last stack_frame_save_area in this stack_segment}
    sfsa := trapped_sfsa{terminate_environment};
    WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
      sfsa := sfsa^.psa_register_image;
    WHILEND:
    {set the critical_frame_flag in the last sfsa of the stack_segment}
    sfsa^.frame_descriptor.critical_frame_flag;
    IFEND;
  IFEND;
  pass trapped_state{normal + trapped_sfsa};
PROCESS_END;1
```

Cont'd →

```
dispose_with_no_handler : PROCESS 5.2.3.6.2 =
  executing_ring_number := p_register.ring_number;
  IF (executing_ring_number = max_user_ring) OR
    (condition is not a continued_condition) THEN
    pass condition_environment;
  ELSE
    {add the condition to the continued_conditions list}
    {and establish a ring alarm}
    continued_condition.condition := condition;
    IF condition is debug THEN
      continued_condition.continued_debug.debug_sfsa :=
        condition_sfsa^;
      continued_condition.continued_debug.debug_index :=
        condition_info^;

    IFEND;
    {establish a ring alarm}
    set established_ring_alarms[executing_ring_number] := true;
    {find the last stack_frame_save_area in this stack_segment}
    sfsa := trapped_sfsa;
    WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
      sfsa := sfsa^.psa_register_image;
    WHILEND;
    {set the critical_frame_flag in the last sfsa of the stack_segment}
    sfsa^.frame_descriptor.critical_frame_flag;
    pass trapped_state{trapped_sfsa + normal status};
  IFEND;
PROCESS_END;1
dispose_with_no_handler : PROCESS 5.2.4.3.2 =
  executing_ring_number := p_register.ring_number;
  IF (executing_ring_number = max_user_ring) OR
    (condition is not a continued_condition) THEN
    pass condition_environment;
  ELSE
    {add the condition to the continued_conditions list}
    {and establish a ring alarm}
    continued_condition.condition := condition;
    IF condition is debug THEN
      continued_condition.continued_debug.debug_sfsa :=
        condition_sfsa^;
      continued_condition.continued_debug.debug_index :=
        condition_info^;
    IFEND;
    {establish a ring alarm}
    set established_ring_alarms[executing_ring_number] := true;
    {find the last stack_frame_save_area in this stack_segment}
    sfsa := trapped_sfsa;
    WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
      sfsa := sfsa^.psa_register_image;
    WHILEND;
    {set the critical_frame_flag in the last sfsa of the stack_segment}
    sfsa^.frame_descriptor.critical_frame_flag;
    pass condition_state{condition_sfsa + normal status};
  IFEND;
PROCESS_END;1
```

Cont'd →

```
dispose_with_handler : PROCESS 5.2.3.6.3 =
  IF (condition is a (primary_condition OR secondary_condition)) AND
    (the corresponding established_descriptor^.condition_handler_active
    condition is set) THEN
    pass condition_environment{system_default};
  ELSE
    IF (condition is a (primary_condition OR secondary_condition)) THEN
      set the corresponding condition in established_descriptor^.
        condition_handler_active;
    IFEND;
    current_condition.handler_active := true;
    current_condition.condition := condition;
    current_condition.condition_info := condition_info;
    current_condition.condition_sfsa := condition_sfsa;
    current_condition.established_descriptor := established_descriptor;
    current_condition.established_descriptor_sfsa :=
      established_descriptor_sfsa;
    enable trap interrupts  {te_register.tef};
    {established_descriptor^.condition_handler points to the}
   {condition handler which will process the condition_handler_request.}
    issue condition_handler_request{condition+condition_info+condition_sfsa};
    disable trap interrupts{te_register.tef};
    current_condition.handler_active := false;
    IF (condition is a (primary_condition OR secondary_condition)) THEN
      clear the corresponding condition in established_descriptor^.
        condition_handler_active;
    IFEND;
    pass trapped_state{trapped_sfsa + condition_handler_response{status}};
  IFEND;
PROCESS_END;1
dispose_with_handler : PROCESS 5.2.4.3.3 =
  current_condition.handler_active := true;
  current_condition.condition := condition;
  current_condition.condition_info := condition_info;
  current_condition.condition_sfsa := condition_sfsa;
  current_condition.established_descriptor := established_descriptor;
  current_condition.established_descriptor_sfsa :=
    established_descriptor_sfsa;
  {established_descriptor^.condition_handler points to the}
  {condition handler which will process the condition_handler_request.}
  issue condition_handler_request{condition + condition_info + condition_sfsa};
  current_condition.handler_active := false;
  pass condition_state{condition_sfsa + condition_handler_response{status}};
PROCESS_END;1
```

Cont'd

```
dmp$cause_condition : PROCESS 5.2.4.1 =
  condition_sfsa := psa_register;
  {find an established_descriptor for the condition}
  sfsa := condition_sfsa;
  REPEAT
    IF (sfsa^.frame_descriptor.on_condition_flag is set) THEN
      established_descriptor := head_condition_list;
      REPEAT
        IF (established_descriptor^.established_boolean = true) AND
          ((established_descriptor^.condition = condition) OR
          (established_descriptor^.condition = all_conditions)) THEN
          established_descriptor is found;
        ELSE
          established_descriptor := established_descriptor^.
            established_descriptor;
        IFEND;
      UNTIL (established_descriptor is found) OR
        (established_descriptor = nil);
    IFEND;
    IF established_descriptor is not found THEN
      sfsa := sfsa^.psa_register_image;
    IFEND;
  UNTIL (established_descriptor is found) OR
    (sfsa^.psa_register_image.ring_number <> executing_ring_number);
  IF (established_descriptor is found) THEN
    local_condition_environment := user_defined_condition + condition_info +
      condition_sfsa;
    pass local_condition_environment;
  ELSE
    set status to abnormal{no established condition handler};
    pass status{cause_user_condition_response};
  IFEND;
  PROCESS_END;1
```

Cont'd →

```
find_condition_handler : PROCESS 5.2.4.3.1 =
  sfsa := condition_sfsa;
  executing_ring_number := p_register.ring_number;
  REPEAT
    IF sfsa^.frame_descriptor.cr_condition_flag is set THEN
      established_descriptor := head_condition_list;
      REPEAT
        IF (established_descriptor^.established_boolean = true) AND
          ((established_descriptor^.condition = condition) OR
          (established_descriptor^.condition = all_conditions)) THEN
          condition_handler is found;
          established_descriptor_sfsa := sfsa;
          pass cond_handler_environment{condition_environment +
            established_descriptor + established_descriptor_sfsa};
        ELSE
          established_descriptor := established_descriptor^.
            established_descriptor_stack;
        IFEND;
      UNTIL (condition_handler is found) OR (established_descriptor = nil);
    IFEND;
    IF condition_handler is not found THEN
      sfsa := sfsa^.psa_register_image;
    IFEND;
  UNTIL (condition_handler is found) OR
    (sfsa^.psa_register_image.ring_number <> executing_ring_number);
  IF (condition_handler is not found) THEN
    pass condition_environment;
  IFEND;
PROCESS_END;1
osp$cause_condition : PROCESS 5.2.4.5 =
  condition_sfsa := psa_register;
  condition := (job_resource_condition | access_method);
  condition_info := nil;
  pass local_condition_environment{condition + condition_info +
      condition_sfsa};
PROCESS_END;1
osp$cause_external_condition : PROCESS 5.2.5.1 =
  form a condition_signal from destination_task_id, executing_task_id, and
    condition;
  issue a signal_request {condition_signal};
  status := signal_response;
  pass status;
PROCESS_END;1
```

Cont'd →

```
enqueue_waiting_task : PROCESS 5.3.3.2.3 =
  IF (number of waiting_tasks in queue_control_table.
    queue_control_block[qid].queue_wait_list <>
    queue_name_table.maximum_queued_items) THEN
    add executing[task_id[waiting_task} to queue_control_table.
  queue_control_block[qid].queue_wait_list;
    clear queue_control_block[qid].queue_control_block_lock;
    issue task_status_change[wait, executing_task_id};
  ELSE
    clear queue_control_block[qid].queue_control_block_lock;
    set status abnormal[maximum number of waiting tasks exceeded};
    pass status;
  IFEND;
PROCESS_END:1
find_condition_handler : PROCESS 5.2.3.6.1 =
  sfsa := trapped_sfsa;
  executing_ring_number := p_register.ring_number;
  REPEAT
    IF sfsa^.frame_descriptor.cr_condition_flag is set THEN
      established_descriptor := head_condition_list;
      REPEAT
        IF (established_descriptor^.established_boolean = true) AND
          ((established_descriptor^.condition = condition) OR
          (established_descriptor^.condition = all_conditions)) THEN
          condition_handler is found;
          established_descriptor_sfsa := sfsa;
          pass cond_handler_environment[condition_environment +
            established_descriptor + established_descriptor_sfsa};
        ELSE
          established_descriptor := established_descriptor^.
            established_descriptor_stack;
        IFEND;
      UNTIL (condition_handler is found) OR (established_descriptor = nil};
    IFEND;
    IF condition_handler is not found THEN
      sfsa := sfsa^.psa_register_image;
    IFEND;
  UNTIL (condition_handler is found) OR
    (sfsa^.psa_register_image.ring_number <> executing_ring_number};
  IF (condition_handler is not found) THEN
    pass condition_environment;
  IFEND;
  PROCESS_END:1
```

Cont'd →

Cont'd

```
enqueue_message : PROCCESS 5.3.3.1.2 =
  qid := queue_identifier;
  {form a message_block}
  message_block.sender_id := executing_task_id;
  message_block_sender_ring := caller_id_ring_number;
  message_block.segment_offset := queue_message.segment_offset;
  message_block.message_type := queue_message.message_type;
  CASE message_block.message_type OF
  =constant_message=
   message_block.constant_message := queue_message.constant_message;
  =pointer_message=
   message_block.pointer_message := queue_message.pointer_message;
  =passed_segment=
   {inform manage_segments of the presence of passed segment(s)}
   {in the job and to delete the passed segment(s) from the}
   {caller's process segment table (address space).}
   message_block.passed_segments.number_of_segments :=
     queue_message.passed_segment_message.number_of_segments;
   FOR i := 1 TO queue_message.passed_segments.number_of_segments DO
     issue system_segment_request(passed, queue_message.
       passed_message_segments.passed_message_segment[i]);
     message_block.passed_segments.passed_segment[i] :=
       system_segment_response;
     {NOTE: manage_segments may return abnormal status as part of}
     {system_segment_response - in which case the FOR loop process}
     {must be reversed to remove passed segments from the job and }
     {add them back into the caller's address space.  The abnormal}
     {status must then be passed to the caller(send_to_queue_resp)}
   FOREND;
  =shared_segment=
   {inform manage_segments of the presence of shared segment(s)}
   {in the job.}
   message_block.shared_segments.number_of_segments :=
     queue_message.shared_segment_message.number_of_segments;
   FOR i := 1 TO queue_message.shared_segments.number_of_segments DO
     issue system_segment_request(shared, queue_message.
       shared_message_segments.shared_message_segment[i]);
     message_block.shared_segments.shared_segment[i] :=
       system_segment_response;
     {NOTE: manage_segments may return abnormal status as part of}
     {system_segment_response - in which case the FOR loop process}
     {must be reversed to remove shared segments from the job. The}
     {abnormal status (send_to_queue_response) must then be passed}
     {to the caller.}
   FOREND;
  CASEND;
  enqueue the message_block on queue_control_table.
    queue_control_block[qid].message_queue;
PROCESS_END:1
```

Cont'd →

Cont'd


```
pmp$connect_queue : PROCESS 5.3.2.1 =
  get caller_id_ring_number from x0 left:
  set queue_name_table_lock;
  qid := 0;
  REPEAT
    IF (queue_name_table.queue_definition[qid].defined_boolean = true) AND
      (queue_name = queue_name_table.queue_definition[qid].queue_name) THEN
      queue_name is in queue_name_table;
    ELSE
      qid := qid + 1;
    IFEND;
  UNTIL (queue_name is in queue_name_table) OR (qid = queue_name_table.
    maximum_queues_per_job);
  IF queue_name is in queue_name_table THEN
    set queue_control_block[qid].queue_control_block_lock;
    clear queue_name_table_lock;
    IF (caller_id_ring_number <= queue_name_table.queue_definition[qid].
      usage_bracket) THEN
      count the number of connected_tasks;
      IF (number of connected_tasks <>
        queue_name_table.maximum_number_connections_per_queue) THEN
        IF (executing_task_id(queue_connected_task) not in
          queue_control_table.queue_control_block[qid].queue_connected_tasks)
          THEN
          (connect the task to the queue)
          add a queue_connected_task(executing_task_id) to queue_control_table.
            queue_control_block[qid].queue_connected_tasks;
          clear queue_control_block[qid].queue_control_block_lock;
          pass queue_identifier = qid;
          pass status(normal);
        ELSE
          clear queue_control_block[qid].queue_control_block_lock;
          set status abnormal(task already connected to this queue);
          pass status;
        IFEND;
      ELSE
        clear queue_control_block[qid].queue_control_block_lock;
        set status abnormal(maximum number of tasks already connected);
        pass status;
      IFEND;
    ELSE
      clear queue_control_block[qid].queue_control_block_lock;
      set status abnormal(caller cannot connect to this queue - callers ring
        (is > usage bracket);
      pass status;
    IFEND;
  ELSE
    clear queue_name_table_lock;
    set status abnormal(no such queue(queue_name) defined);
    pass status;
  IFEND;
PROCESS_END;1
```

Cont'd →

```
pmp$continue_to_cause : PROCESS 5.2.4.4 =
  IF current_condition.handler_active THEN
    {find an established_descriptor for the condition}
    established_descriptor := current_condition.established_descriptor^.
      established_descriptor_stack;
    condition := current_condition.condition;
    WHILE (established_descriptor not found) AND (established_descriptor
      <> nil) DO
      IF (established_descriptor^.established_boolean = true) AND
        ((established_descriptor^.condition = condition) OR
        (established_descriptor^.condition = all_conditions)) THEN
        established_descriptor is found;
      ELSE
        established_descriptor := established_descriptor^.
          established_descriptor_stack;
      IFEND;
    WHILEND;
    {last stack_frame to search has a psa_register_image = nil}
    sfsa := current_condition.established_descriptor_sfsa^.psa_register_image;
    WHILE (established_descriptor is not found) AND (sfsa <> nil) DO
      IF sfsa^.frame_descriptor.on_condition_flag is set THEN
        established_descriptor := head_condition_list;
        REPEAT
          IF (established_descriptor^.established_boolean = true) AND
            ((established_descriptor^.condition = condition) OR
            (established_descriptor^.condition = all_conditions)) THEN
            established_descriptor is found;
          ELSE
            established_descriptor := established_descriptor^.
              established_descriptor_stack;
          IFEND;
        UNTIL (established_descriptor is found) OR (established_descriptor =
          nil);
      IFEND;
      IF established_descriptor is not found THEN
        sfsa := sfsa^.psa_register_image;
      IFEND;
    WHILEND;
    IF established_descriptor is found THEN
      established_descriptor_sfsa := sfsa;
      condition_info := current_condition.condition_info;
      condition_sfsa := current_condition.condition_sfsa;
      pass disposable_cond_environment(condition + condition_info +
        condition_sfsa + established_descriptor + established_descriptro_sfsa);
      pass status(normal);
    ELSE
      pass status(abnormal - no established descriptor found);
    IFEND;
  ELSE
    pass status(abnormal - continue_to_cause is invalid from other than
      a condition handler);
  IFEND;
PROCESS_END;1
```

Cont'd →

```
pmp$define_queue : PROCESS 5.3.1.1 =
  get caller_id_ring_number from x0 left;
  IF caller_id_ring_number <= removal_bracket THEN
    set queue_name_table_lock;
    qid := 0;
    REPEAT
      IF (queue_name_table.queue_definition[qid].defined_boolean = true) AND
        (queue_name = queue_name_table.queue_definition[qid].queue_name) THEN
        queue_name is in queue_name_table;
      ELSE
        qid := qid + 1;
      IFEND;
    UNTIL (queue_name is in queue_name_table) OR (qid = queue_name_table.
      maximum_queues_per_job);
    IF queue_name is not in queue_name_table THEN
      /search_for_free_entry/
      FOR free_entry := 0 TO queue_name_table.maximum_queues_per_job DO
        IF (queue_name_table.queue_definition[free_entry].defined_boolean =
          false) THEN
          exit /search_for_free_entry/;
        IFEND;
      FOREND;
      queue_name_table.queue_definition[free_entry].queue_name := queue_name;
      queue_name_table.queue_definition[free_entry].removal_bracket :=
        removal_bracket;
      queue_name_table.queue_definition[free_entry].usage_bracket :=
        usage_bracket;
      queue_name_table.queue_definition[free_entry].defined_boolean := true;
      set queue_control_block[free_entry].queue_control_block_lock;
      clear queue_name_table_lock;
      initialize queue_control_table[free_entry];
      {queue_connected_tasks are probably really a double linked list}
      {queued_items are probably single linked QUEUES - all will require}
      {initialized to a null state.}
      clear queue_control_block[free_entry].queue_control_block_lock;
      pass status{normal};
    ELSE
      clear queue_name_table_lock;
      set status abnormal{maximum queues defined};
      pass status;
    IFEND;
  ELSE
    set status abnormal{removal_bracket > caller execution ring};
    pass status;
  IFEND;
PROCESS_END;1
```

Cont'd →

```
pmp$disconnect_queue : PROCESS 5.3.2.2 =
  IF (qid <= queue_name_table.maximum_queues_per_job) AND
    (queue_name_table.queue_definition.defined_boolean = true) THEN
    set queue_control_block[qid].queue_control_block_lock;
    IF (executing_task_id[queue_connected_task] in
      queue_control_table.queue_control_block[qid].queue_connected_tasks) THEN
      {disconnect the task from the queue}
      delete executing_task_id[queue_connected_task] from
        queue_control_table.queue_control_block[qid].queue_connected_tasks;
      IF (executing_task_id[waiting_task] in
        queue_control_table.queue_control_block[qid].queue_wait_list) THEN
        delete executing_task_id[waiting_task] from
          queue_control_table.queue_control_block[qid].queue_wait_list;
      IFEND;
      clear queue_control_block[qid].queue_control_block_lock;
      pass status[normal];
    ELSE
      clear queue_control_block[qid].queue_control_block_lock;
      set status abnormal[task is not connected to this queue];
      pass status;
    IFEND;
  ELSE
    set status abnormal[illegal qid or queue is not defined];
    pass status;
  IFEND;
PROCESS_END;1
pmp$disestablish_cond_handler : PROCESS 5.2.2 =
  {find an established_descriptor for the condition}
  sfsa := psa_register;
  REPEAT
    IF sfsa^.frame_descriptor.cr_condition_flag is set THEN
      established_descriptor := head_condition_list;
      REPEAT
        IF (established_descriptor^.established_boolean = true) AND
          (established_descriptor^.condition = condition) THEN
          established_descriptor is found;
        ELSE
          established_descriptor := established_descriptor^.
            established_descriptor_stack;
        IFEND;
      UNTIL (established_descriptor is found) OR (established_descriptor = nil);
    IFEND;
    IF established_descriptor is not found THEN
      sfsa := sfsa^.psa_register_image;
    IFEND;
  UNTIL (established_descriptor is found) OR
    (stack_segment[max_user_ring] has been searched);
  IF (established_descriptor is found) THEN
    {disestablish the established descriptor}
    established_descriptor^.established_boolean := false;
    pass status[normal];
  ELSE
    pass status[abnormal - no descriptor found];
  IFEND;
PROCESS_END;1
```

Cont'd →

```
pmp$establish_condition_handler : PROCESS 5.2.1 =
  {ensure establish_descriptor is in caller's stack_frame}
  sfsa := psa_register;
  IF (established_descriptor < sfsa^.dsp_register_image) AND
    (established_descriptor > sfsa^.csf_register_image) THEN
    disable trap interrupts(te_register.tef);
    {stack the established_descriptor on established_condition_stack}
    IF (sfsa^.frame_descriptor.on_condition_flag is set) THEN
      established_descriptor^.established_descriptor_stack :=
        head_condition_list;
    ELSE
      established_descriptor^.established_descriptor_stack := nil;
    IFEND;
    head_condition_list := established_descriptor;
    established_descriptor^.condition := condition;
    established_descriptor^.established_boolean := true;
    enable trap interrupts(te_register.tef);
    pass status(normal);
  ELSE
    pass status(abnormal - descriptor not caller's stack frame);
  IFEND;
PROCESS_END;1
pmp$get_queue_limits : PROCESS 5.3.4 =
  queue_limits := queue_name_table.maximum_queues_per_job +
                  queue_name_table.maximum_connections_per_queue +
                  queue_name_table.maximum_queued_items;
  pass queue_limits;
PROCESS_END;1
```

Cont'd →

```
pmp$remove_queue-1-PROCESS-5.3.1.2-=---------
  get caller_id_ring_number from x0 left;
  set queue_name_table_lock;
  qid := 0;
  REPEAT
    IF (queue_name_table.queue_definition[qid].defined_boolean = true) AND
      (queue_name = queue_name_table.queue_definition[qid].queue_name) THEN
      queue_name is in queue_name_table;
    ELSE
      qid := qid + 1;
    IFEND;
  UNTIL (queue_name is in queue_name_table) OR (qid = queue_name_table.
    maximum_queues_per_job);
  IF queue_name is in queue_name_table THEN
    IF (caller_id_ring_number <= queue_name_table.queue_definition[qid].
      removal_bracket) THEN
      IF queue_control_table.queue_control_block[qid].queue_connected_tasks
        is empty THEN
        IF queue_control_table.queue_control_block[qid].message_queue
          is empty THEN
          {remove the queue definition}
          queue_name_table.queue_definition[qid].defined_boolean := false;
          clear queue_name_table_lock;
          pass status[normal];
        ELSE
          clear queue_name_table_lock;
          set status abnormal{there messages enqueued};
          pass status;
        IFEND;
      ELSE
        clear queue_name_table_lock;
        set status abnormal{tasks are still connected to queue};
        pass status;
      IFEND;
    ELSE
      clear queue_name_table_lock;
      set status abnormal{caller cannot remove this queue - callers ring
        {is > removal bracket};
      pass status;
    IFEND;
  ELSE
    clear queue_name_table_lock;
    set status abnormal{no such queue defined};
    pass status;
  IFEND;
PROCESS_END;1
```

Cont'd →

```
pmp$status_queue : PROCESS 5.3.2.3 =
  IF (qid <= queue_name_table.maximum_queues_per_job) THEN
    IF (queue_name_table.queue_definition[qid].defined_boolean = true) THEN
      number_connected_tasks := number of queue_connected_task in
        queue_control_table.queue_control_block[qid].queue_connected_tasks;
      number_queued_messages := number of message_blocks in
        queue_control_table_queue_control_block[qid].message_queue;
      number_waiting_taskss := number of waiting_tasks in
        queue_control_table_queue_control_block[qid].queue_wait_list;
      pass status_queue_response{number_connected_tasks + number_queued_items};
      pass status{normal};
    ELSE
      set status abnormal{queue(qid) not defined};
      pass status;
    IFEND;
  ELSE
    set status abnormal{illegal qid};
    pass status;
  IFEND;
PROCESS_END;1
pmp$status_queues_defined : PROCESS 5.3.1.3 =
  number_defined_queues := 0;
  FOR qid := 0 TO queue_name_table.maximum_queues_per_job - 1 DO
    IF queue_name_table.queue_definition[qid].defined_boolean = true THEN
      number_defined_queues := number_defined_queues + 1;
    IFEND;
  FOREND;
  pass number_defined_queues;
PROCESS_END;1
pmp$test_condition_handler : PROCESS 5.2.4.2 =
  {test_sfsa_pointer points to a dummy stack_frame_save_area supplied by}
  {the caller.}
  test_sfsa_pointer^.psa_register_image := psa_register;
  local_condition_environment := condition_to_test + condition_info +
    test_sfsa_pointer;
  pass local_condition_environment;
PROCESS_END;1
ready_waiting_tasks : PROCESS 5.3.3.1.3 =
  WHILE (there are waiting_tasks in queue_control_table.queue_control_block[qid].
      queue_wait_list) DO
    issue task_status_change{ready, queue_control_table.
       queue_control_block[qid].queue_wait_list};
    delete the waiting_task from the queue_control_table.
       queue_control_block[qid].queue_wait_list;
  WHILEND;
  clear queue_control_block[qid].queue_control_block_lock;
PROCESS_END;1
```

Cont'd →

```
reestablish_trapped_environment : PROCESS 5.2.3.2 =
  IF status is normal THEN
    IF (wait_activity_list[executing_task_id.local_task_id] is empty) THEN
      enable trap interrupts[te_register.tef];
      issue task_status_change[wait];
    ELSE
      set trap enable delay and trap enable flip-flop in te_register;
    IFEND;
  ESLE
    issue terminate_xtask_request[executing_task_id + status];
    IF terminate_xtask_response = establish_deferred_exit THEN
      {establish a ring alarm for deferred exit}
      executing_ring_number := p_register.ring_number;
      deferred_exit.ring_number := executing_ring_number;
      set established_ring_alrams[executing_ring_number] := true;
      {find the last stack_frame_save_area in this stack_segment}
      sfsa := trapped_sfsa;
      WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
        sfsa := sfsa^.psa_register_image;
      WHILEND;
      {set the critical_frame_flag in the last sfsa of the stack_segment}
      sfsa^.frame_descriptor.critical_frame_flag;
    IFEND;
      set trap enable delay and trap enable flip-flop in te_register;
  IFEND;
PROCESS_END;1
route_signals : PROCESS 5.2.3.4.4 =
  WHILE the signal_buffer is not empty DO
    dequeue a signal from the signal_buffer;
    CASE signal.signal_identifier OF
    =terminate_signal_id=
     pass terminate_environment[signal + trapped_sfsa];
    =callend_signal_id=
     pass callend_environment[signal + trapped_sfsa];
    =condition_signal_id=
     pass condition_signal_erviron[signal + trapped_sfsa];
    CASEND;
  WHILEND;
PROCESS_END;1
```

Cont'd →

```
route_traps : PROCESS 5.2.3.1.7 =------
  trapped_sfsa := psa_register;
  traps := (trapped_sfsa^.user_condition_register image AND
    trapped_sfsa^.user_mask);
  {traps is equivalent to secondary_conditions}
  WHILE traps is not empty DO
    IF (process_interval_timer is in traps) THEN
      traps := traps XOR process_interval_timer;
      pass pit_environment{trapped_sfsa};
    IFEND;
    IF (free_flag is in traps) THEN
      traps := traps XOR free_flag;
      pass free_flag_environment{trapped_sfsa};
    IFEND;
    IF (critical_frame_flag is in traps) THEN
      traps := traps XOR critical_frame_flag;
      pass cff_environment{trapped_sfsa};
    IFEND;
    IF (debug is in traps) THEN
      traps := traps XOR debug;
      pass debug_environment{trapped_sfsa};
    IFEND;
    IF (keypoint is in traps) THEN
      traps := traps XOR keypoint;
    IFEND;
    FOR other_secondary_condition := privileged_instruction_fault TO
      invalid_bdp_data DO
      IF (other_secondary_condition is in traps) THEN
        traps := traps XOR other_secondary_condition;
        pass other_trap_environment{other_secondary_condition + trapped_sfsa};
      IFEND;
    FOREND;
  WHILEND;
PROCESS_END;1
set_task_disposition : PROCESS 5.2.4.3.5 =
  IF status is not normal THEN
    issue terminate_xtask_request{status};
    IF terminate_xtask_response = establish_deferred_exit THEN
      {establish a ring alarm for deferred exit}
    executing_ring_number := p_register.ring_number;
      deferred_exit.ring_number := executing_ring_number;
      set established_ring_alarms[executing_ring_number] := true;
      {find the last stack_frame_save_area in this stack_segment}
      sfsa := condition_sfsa;
      WHILE sfsa^.psa_register_image.ring_number = executing_ring_number DO
        sfsa := sfsa^.psa_register_image;
      WHILEND;
      {set the critical_frame_flag in the last sfsa of the stack_segment}
      sfsa^.frame_descriptor.critical_frame_flag;
    IFEND;
  IFEND;
PROCESS_END;1
```

Cont'd →

```
syp$cause_condition : PROCESS 5.2.5.3 =
  get the execution_control_block address of the destination_task_id
    from the primary_task_list;
  get the signal_buffer address of the destination_task_id from the
    primary_task_list;
  form a condition_signal from condition, destination_task_id, and
    source_task_id;
  enqueue the condition_signal on the signal_buffer;
  {i.e., pass the condition_signal}
  set the free_flag in the exchange_package of the execution_control_block;
  issue task_status_change{destination_task_id, ready};
PROCESS_END;1
system_default_disposers : PROCESS 5.2.3.6.4 =
  {general process only - system default disposers are not completely}
  {defined for all conditions}
  CASE condition OF
    =primary_conditions, secondary_conditions=
     set status to abnormal with the appropriate status condition indicator
       set;
    =access_method_condition=
     NOP or set status to abnormal with the appropriate status condition
       indicator set;
    =job_resource_condition=
     set status to abnormal with the appropriate status condition indicator
       set;
    =segment_access_condition=
     set status to abnormal with the appropriate status condition indicator
       set;
  CASEND;
  pass trapped_state{trapped_sfsa + status};
      {abnormal status will cause the task to terminate}
PROCESS_END;1
system_default_disposers : PROCESS 5.2.4.3.4 =
  {general process only - system default disposers are not completely}
  {defined for all conditions}
  CASE condition OF
    =access_method_condition=
     NOP or set status to abnormal with the appropriate status condition
       indicator set;
    =job_resource_condition=
     set status to abnormal with the appropriate status condition indicator
       set;
  CASEND;
  pass condition_state{condition_sfsa + status};
      {abnormal status will cause the task to terminate}
PROCESS_END;1
```

Cont'd →

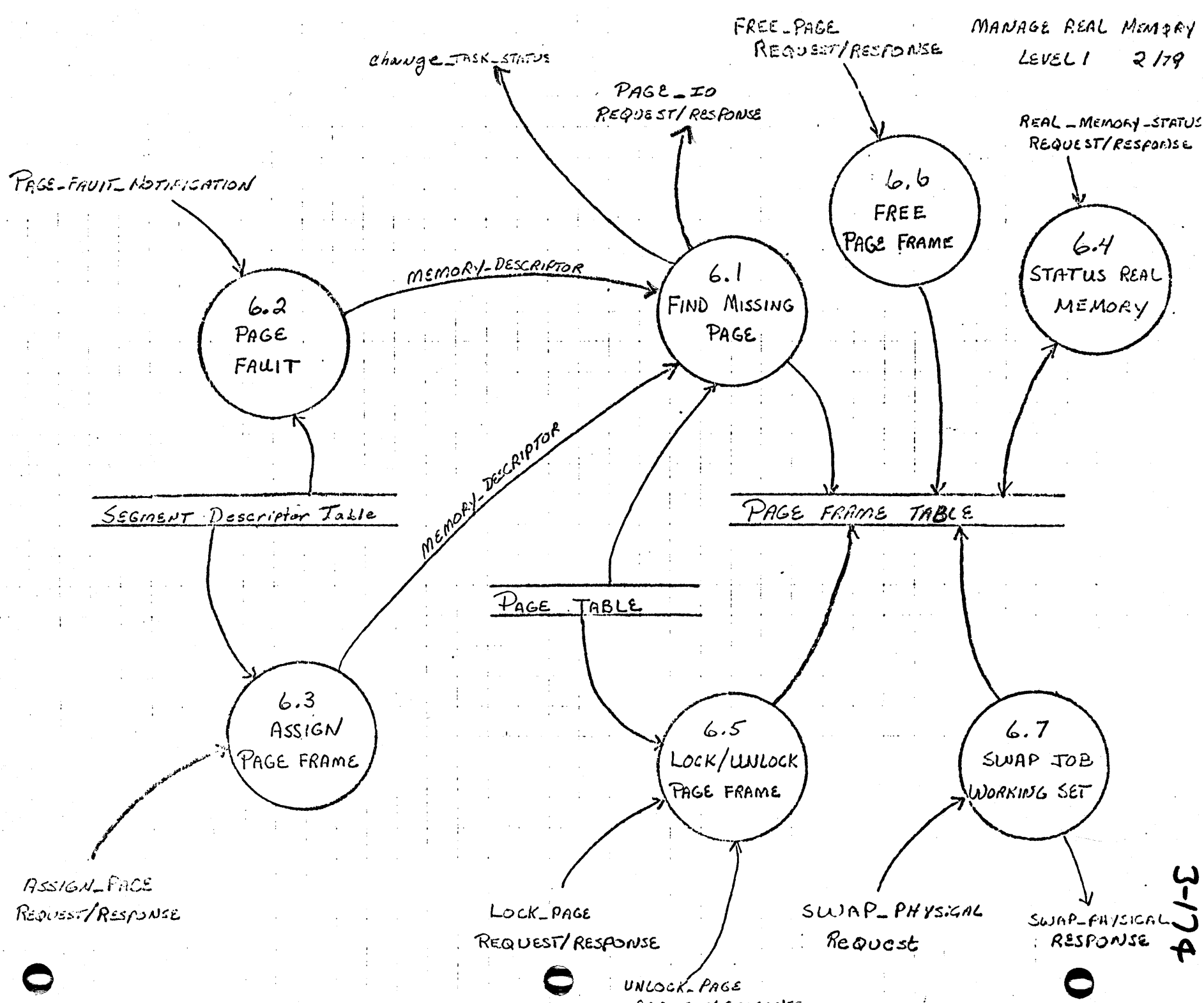
Cont'd

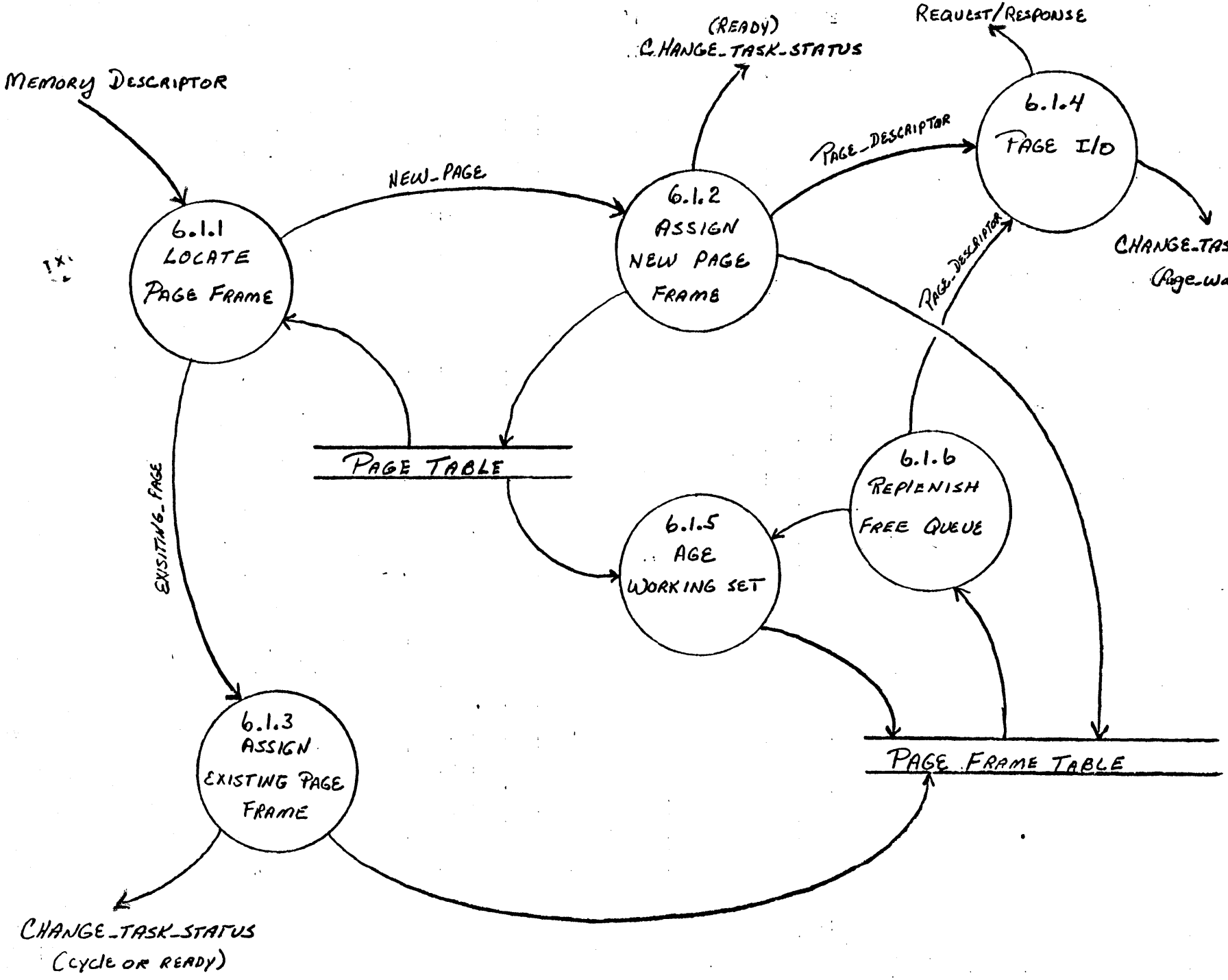
```
validate_queue_access : PROCESS 5.3.3.1.1 =
  get caller_id_ring_number from x0 left;
  IF (qid < queue_name_table.maximum_queues_per_job) THEN
    IF (queue_name_table.queue_definitions[qid].defined_boolean = true) THEN
      IF (caller_id_ring_number <= queue_name_table.queue_definition[qid].
         usage_bracket) THEN
        set queue_control_block[qid].queue_control_block_lock;
        IF (executing_task_id{queue_connected_task} is in
          queue_control_table.queue_control_block[qid].queue_connected_tasks)
          THEN
          IF (number of message_blocks in queue_control_table.
            queue_control_block[qid].message_queue <>
            queue_name_table.maximum_items_per_queue) THEN
            pass caller_id_ring_number;
            pass status{normal};
          ELSE
            clear queue_control_block[qid].queue_control_block_lock;
            set status abnormal{maximum messages exceeded};
            pass status;
          IFEND;
        ELSE
          clear queue_control_block[qid].queue_control_block_lock;
          set status abnormal{task is not connected to this queue(qid)};
          pass status;
        IFEND;
      ELSE
        set status abnormal{caller cannot send to queue - executing ring}
          {number > usage bracket};
        pass status;
      IFEND;
    ELSE
      set status abnormal{queue(qid) is not defined};
      pass status;
    IFEND;
  ELSE
    set status abnormal{qid is illegal};
    pass status;
  IFEND;
PROCESS_END;1
```

Cont'd →

```
validate_queue_access : PROCESS 5.3.3.2.1 =
  get caller_id_ring_number from x0 left;
  IF (qid < queue_name_table.maximum_queues_per_job) THEN
    IF (queue_name_table.queue_definitions[qid].defined_boolean = true) THEN
      IF (caller_id_ring_number <= queue_name_table.queue_definition[qid].
         usage_bracket) THEN
        set queue_control_block[qid].queue_control_block_lock;
        IF (executing_task_id[queue_connected_task] is in
          queue_control_table.queue_control_block[qid].queue_connected_tasks)
          THEN
          pass qid;
          pass status[normal];
        ELSE
          clear queue_control_block[qid].queue_control_block_lock;
          set status abnormal[task is not connected to this queue(qid)];
          pass status;
        IFEND;
      ELSE
        set status abnormal[caller cannot send to queue - executing ring]
          [number > usage bracket];
        pass status;
      IFEND;
    ELSE
      set status abnormal[queue(qid) is not defined];
      pass status;
    IFEND;
  ELSE
    set status abnormal[qid is illegal];
    pass status;
  IFEND;
PROCESS_END;1
verify_signal_destination : PROCESS 5.2.5.2 =
  IF destination_task_id is in the primary_task_list THEN
    transform the condition_signal into a pass_condition_to_task_rec:
    [destination_task_id + source_task_id + condition]
    pass status[normal];
    pass pass_condition_to_task_req;
  ELSE
    pass status[abnormal - invalid destination_task_id];
  IFEND;
PROCESS_END;
```

3.6 MANAGE REAL MEMORY


FREE_PAGE REQUEST/RESPONSE
change_TASK_STATUS
PAGE_ID REQUEST/RESPONSE
REAL_MEMORY_STATUS REQUEST/RESPONSE
PAGE_FAULT_NOTIFICATION
6.6 FREE PAGE FRAME
6.4 STATUS REAL MEMORY
6.2 PAGE FAULT
MEMORY_DESCRIPTOR
6.1 FIND MISSING PAGE
SEGMENT Descriptor Table
MEMORY_DESCRIPTOR
PAGE FRAME TABLE
PAGE TABLE
6.3 ASSIGN PAGE FRAME
6.5 LOCK/UNLOCK PAGE FRAME
6.7 SWAP JOB WORKING SET
ASSIGN_PAGE REQUEST/RESPONSE
LOCK_PAGE REQUEST/RESPONSE
UNLOCK_PAGE REQUEST/RESPONSE
SWAP_PHYSICAL Request
SWAP_PHYSICAL RESPONSE


MEMORY DESCRIPTOR
6.1.1
LOCATE
PAGE FRAME
NEW_PAGE
6.1.2
ASSIGN
NEW PAGE
FRAME
(READY)
CHANGE_TASK_STATUS
PAGE_ID
REQUEST/RESPONSE
6.1.4
PAGE I/O
PAGE_DESCRIPTOR
CHANGE_TASK_STATUS
(Page_wait)
PAGE_DESCRIPTOR
PAGE TABLE
EXISTING_PAGE
6.1.5
AGE
WORKING SET
6.1.6
REPLENISH
FREE QUEUE
6.1.3
ASSIGN
EXISTING PAGE
FRAME
PAGE FRAME TABLE
CHANGE_TASK_STATUS
(cycle OR READY)

```
advise_in_request : FLOW
  = process_virtual_address
    + length;
advise_out_request : FLOW
  = process_virtual_address
    + length
    + (wait
    ! nowait);
assign_page_request : FLOW
  = free_page_request;
change_segment_access_request : FLOW
  = process_virtual_address
    + segment_attributes;
change_segment_length_request : FLOW
  = process_virtual_address
    + maximum_segment_length;
create_segment_request : FLOW
  = segment_attribute
    + segment_number
    + segment_length;
delete_segment_request : FLOW
  = process_virtual_address;
evict_segment_resources_request : FLOW
  = process_virtual_address;
existing_page : FLOW
  = memory_descriptor;
free_page_request : FLOW
  = process_virtual_address
```

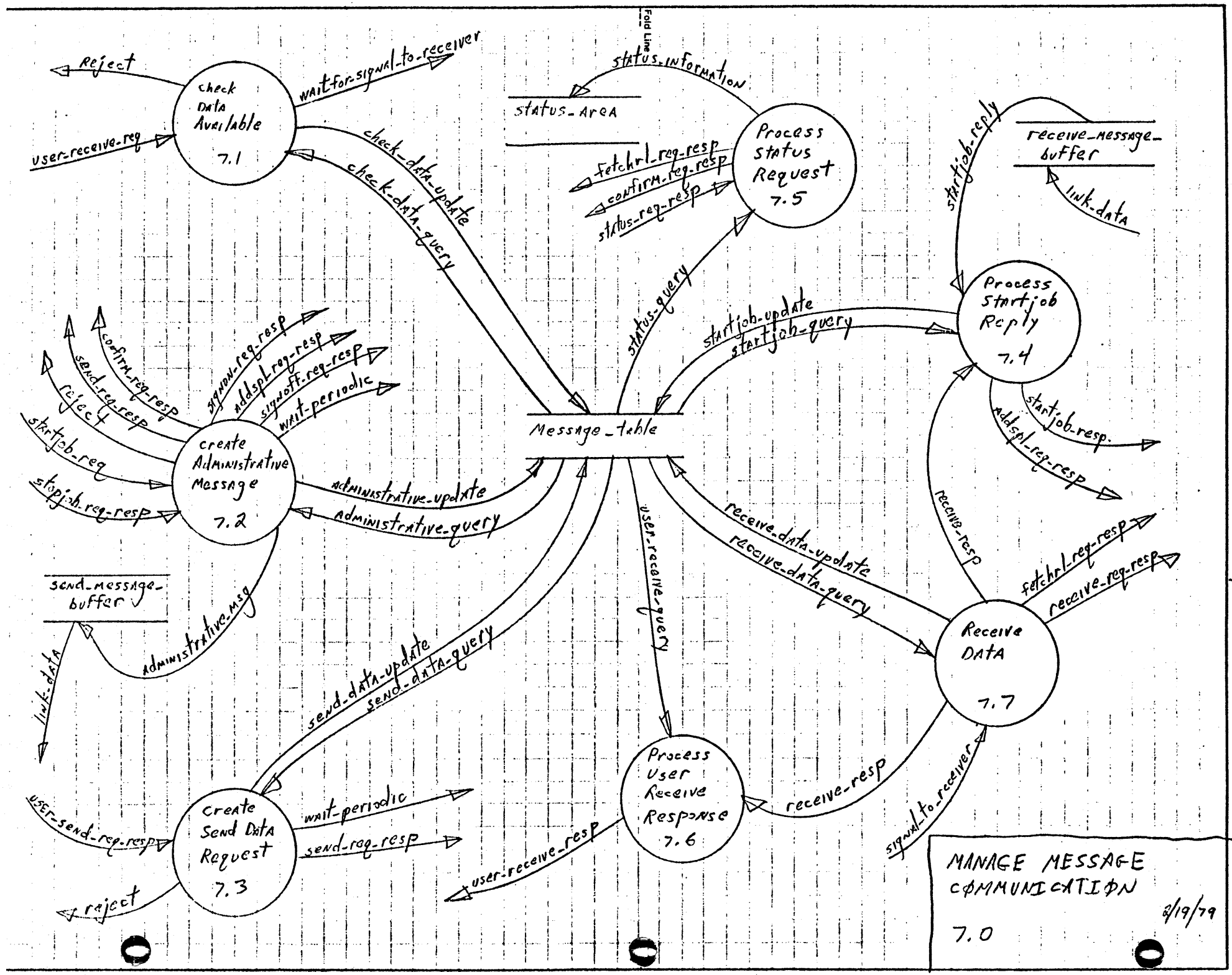
```
    + length;
lock_page_request : FLOW
  = free_page_request;
memory_descriptor : FLOW
  = system_virtual_address
    + taskid
    + number_of_pages;
new_page : FLOW
  = memory_descriptor;
page_descriptor : FLOW
  = system_virtual_address
    + taskid
    + page_frame_table_index;
page_frame_table : FILE
  = <page_age
    + page_table_index
    + active_io_count
    + time_stamp
    + queued_taskid
    + queue_type
    + running_job_ordinal>;
page_id : FILE
  = active_segment_id
    + page_offset;
page_table : FILE
  = <page_id
    + physical_address
    + [valid]
```

OUT 79/02/26. 08.42.40.

```
    + [continue]
    + [used]
    + [modified]>;
process_virtual_address : FLOW
  = ring_number
    + segment_number
    + byte_offset;
queue_type : FILE
  = (free
    ! available_modified
    ! shared_working_set
    ! wired
    ! available
    ! job_working_set);
segment_attributes : FLOW
  = tbd "gives information required to set up a segment descriptor entry";
segment_descriptor_table : FILE
  = <[wired]
    + [shared]
    + [stack]
    + [sequential]
    + [cache_by_pass]
    + [read]
    + [write]
    + [binary]
    + [execute]
    + [local_key]
    + [global_key]
```

```
    + file_id
    + active_segment_id>;
segment_number : FLOW
  = 0..4095;
status_segment_request : FLOW
  = process_virtual_address;
status_segment_response : FLOW
  = maximum_segment_length
    + current_segment_length
    + segment_attributes;
swap_physical_request : FLOW
  = running_job_id;
swap_physical_response : FLOW
  = <real_memory_address
    + system_virtual_address>
    + working_set_size;
swap_working_set_request : FLOW
  = running_job_id;
unlock_page : FLOW
  = free_page_request;
```

3.7 MANAGE MESSAGE COMMUNICATION


Fold Line
Check Data Available 7.1
Process Status Request 7.5
Process Startjob Reply 7.4
Create Administrative Message 7.2
Message_table
Receive DATA 7.7
Process User Receive Response 7.6
Create Send Data Request 7.3
status_area
receive_message_buffer
send_message_buffer
reject
wait-for-signal-to-receiver
user-receive-req
check-data-query
check-data-update
status_information
fetchrl-req-resp
confirm-req-resp
status-req-resp
status-query
startjob-reply
link-data
startjob-update
startjob-query
startjob-resp.
addspl-req-resp
signon-req-resp
addspl-req-resp
signoff-req-resp
wait-periodic
send-req-resp
confirm-req-resp
reject
startjob-req
stopjob-req-resp
administrative-update
administrative-query
receive-data-update
receive-data-query
receive-resp
fetchrl-req-resp
receive-req-resp
user-receive-query
administrative-msg
link-data
send-data-update
send-data-query
wait-periodic
send-req-resp
user-send-req-resp
reject
user-receive-resp
receive-resp
signal-to-receiver
MANAGE MESSAGE COMMUNICATION
7.0
8/19/79
Diagram ID:
Name:
Description:
3-182

```
administrative_msg : FLOW
  = ((pf_descriptor + startjob_msg_type) | stopjob_msg_type);
administrative_query : FLOW
  = message_application_name + exec_application_name
    + message_response_pointer;
administrative_reply : FLOW
  = startjob_reply;
administrative_update : FLOW
  = user_waiting_to_receive + user_waiting_to_send + startjob_sent
    + waiting_to_send_startjob + receive_buffer_location
    + receive_buffer_length;
check_data_query : FLOW
  = message_application_name + connected_application_name;
check_data_update : FLOW
  = user_waiting_to_receive + receive_buffer_location
    + receive_buffer_length;
connected_application_name : FLOW
  = system_name { name of one "user" correspondant task };
exec_application_name : FLOW
  = pre_defined_name { name of message communication exec on 170 };
message_application_name : FLOW
  = system_name { name of one "user" correspondant task };
message_communication_req : FLOW
  = (startjob_req | stopjob_req | status_req | user_receive_req
     | user_send_req);
message_communication_resp : FLOW
  = (startjob_resp | stopjob_resp | status_resp | user_receive_resp
     | user_send_resp);
message_response_pointer : FLOW
  = ;
message_table : FILE
  = message_application_name + connected_application_name
    + exec_application_name { canned name }
    + message_response_pointer + user_waiting_to_receive
    + user_waiting_to_send + startjob_sent
    + waiting_to_send_stopjob + receive_data_available
    + ((receive_buffer_location + receive_buffer_length)
       | (send_buffer_location + send_buffer_length))
    + receive_list;
receive_data_available : ELEMENT
  = true ! false;
receive_data_query : FLOW
  = message_application_name + connected_application_name
    + user_waiting_to_receive + startjob_sent + receive_buffer_location
    + receive_buffer_length;
receive_data_update : FLOW
  = receive_data_available;
receive_message_buffer : FILE
  = administrative_reply;
send_data_query : FLOW
  = message_application_name + connected_application_name
    + message_response_pointer;
send_data_update : FLOW
  = user_waiting_to_send + send_buffer_location
    + send_buffer_length;
send_message_buffer : FILE
  = 0 <administrative_msg> 2;
startjob_query : FLOW
```

```
  = message_application_name + message_response_pointer;
startjob_reply : FLOW
  = connected_application_name;
startjob_req : FLOW
  = pf_descriptor;
startjob_resp : FLOW
  = return_status { tbd };
startjob_sent : ELEMENT
  = true ! false;
startjob_update : FLOW
  = connected_application_name + startjob_sent;
status_area : FILE
  = status_information;
status_information : FLOW
  = c170_job_signed_on + data_to_receive + ok_to_send;
status_query : FLOW
  = message_application_name + connected_application_name;
status_req : FLOW
  = status_area_location;
status_req_resp : FLOW
  = (status_req ! status_resp);
status_resp : FLOW
  = status_information + return_status { tbd };
stopjob_req : FLOW
  = { no data }
  ;
stopjob_req_resp : FLOW
  = (stopjob_req ! stopjob_resp);
stopjob_resp : FLOW
  = return_status { tbd };
transfer_count : ELEMENT
  = ;
user_receive_query : FLOW
  = message_response_pointer;
user_receive_req : FLOW
  = receive_buffer_location + receive_buffer_length;
user_receive_resp : FLOW
  = message_length + transfer_count + message + return_status { tbd };
user_send_req : FLOW
  = send_buffer_location + send_buffer_length;
user_send_req_resp : FLOW
  = (user_send_req ! user_send_resp);
user_send_resp : FLOW
  = return_status { tbd };
user_waiting_to_receive : ELEMENT
  = true ! false;
user_waiting_to_send : ELEMENT
  = true ! false;
waiting_to_send_startjob : ELEMENT
  = true ! false;
```

3.8 MANAGE_PERMANENT_FILE

Diagram 8
Manage Permanent Files
RJT 03/01/79

3-186

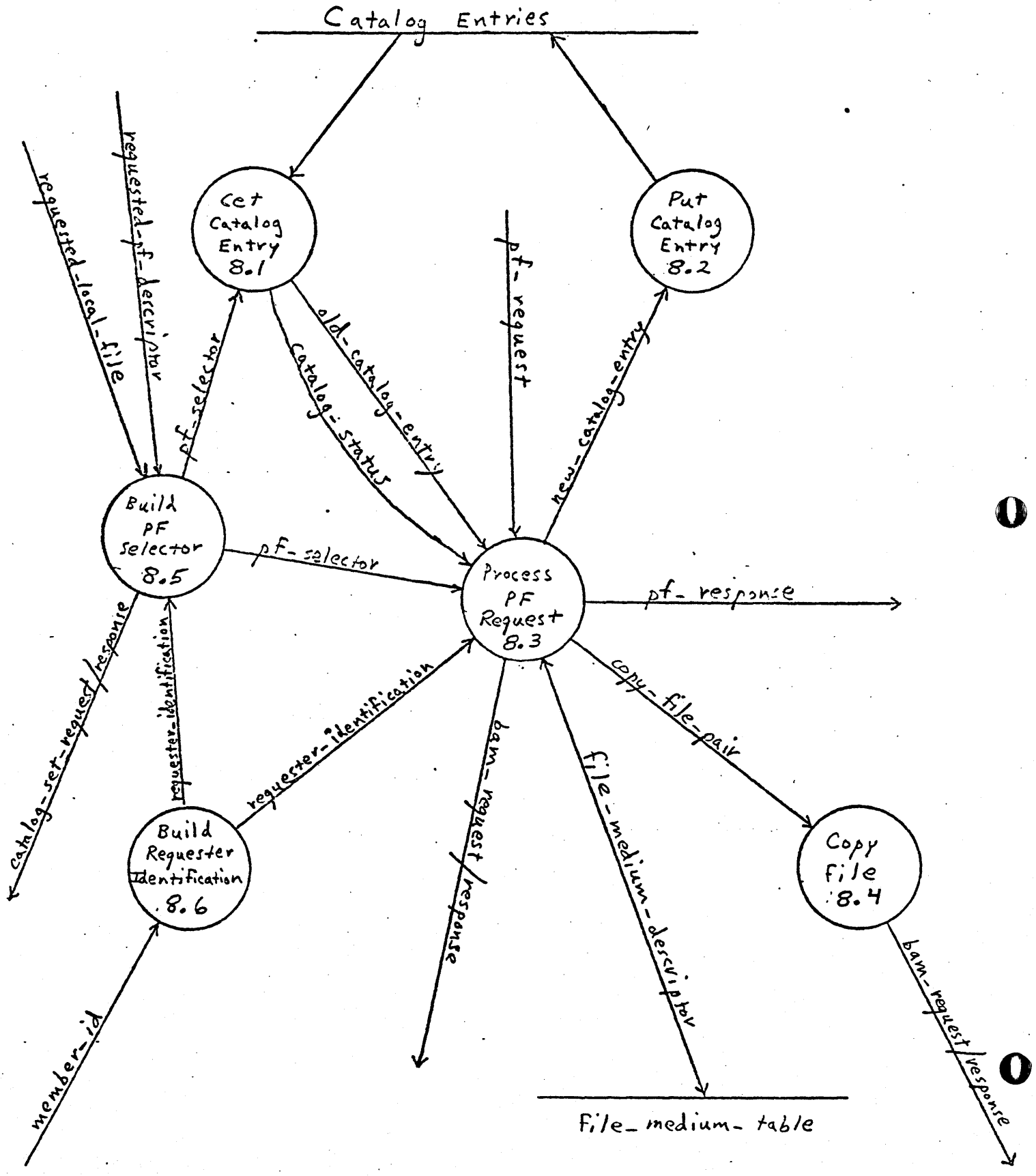


Diagram 8.3
Process Pf Request
RJT 03/01/79

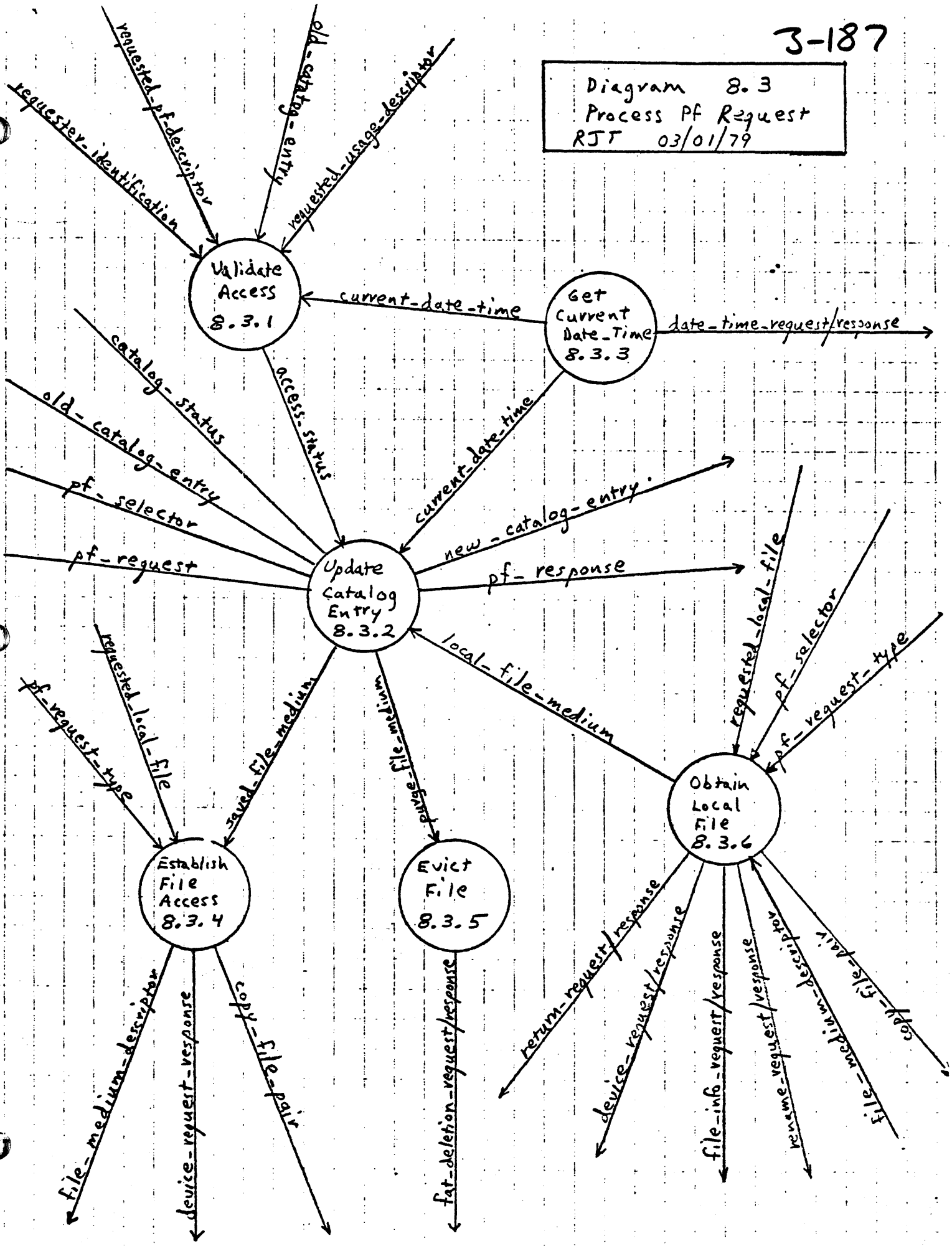


Diagram 8.3.1
Validate Access
RJT 03/01/79

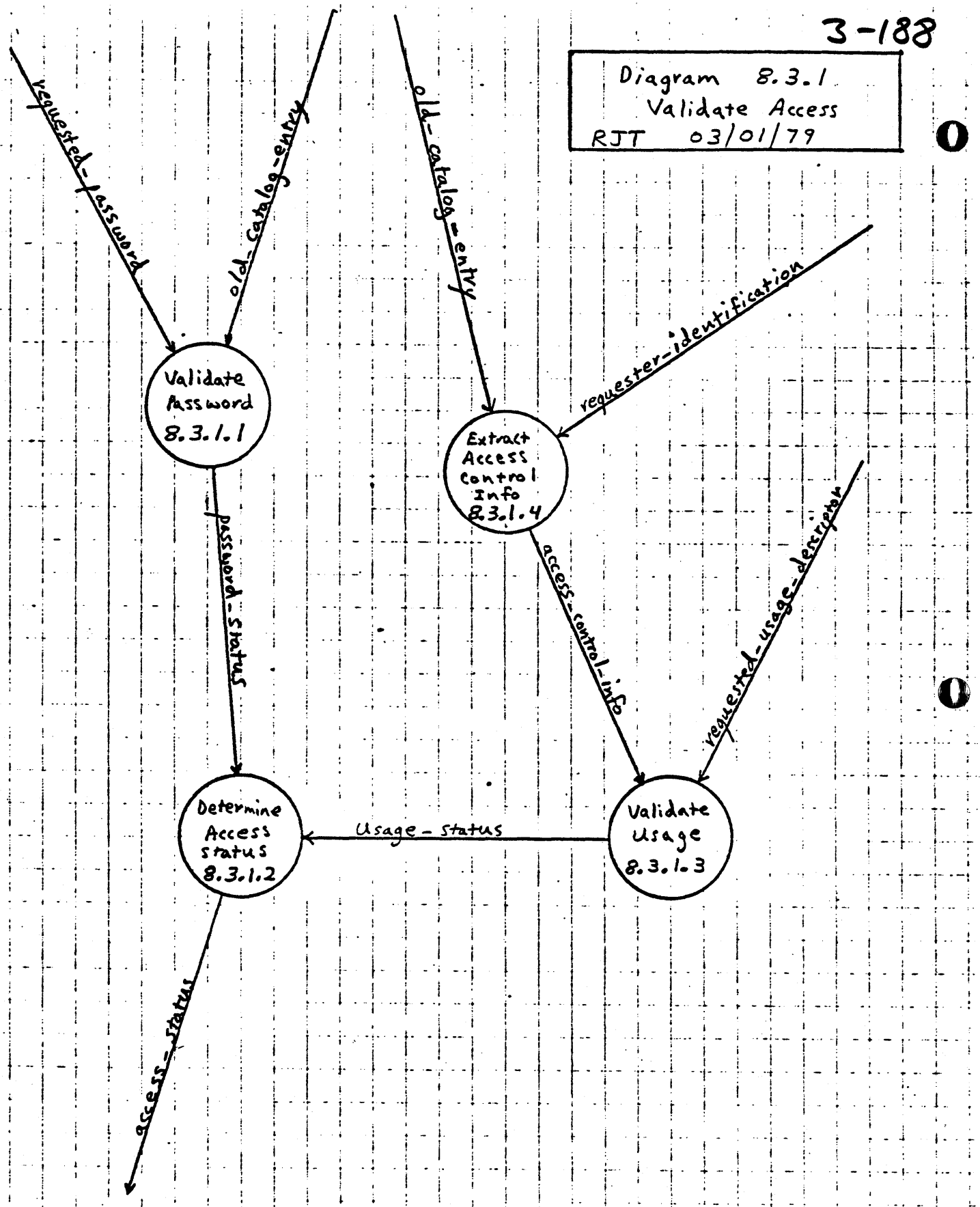
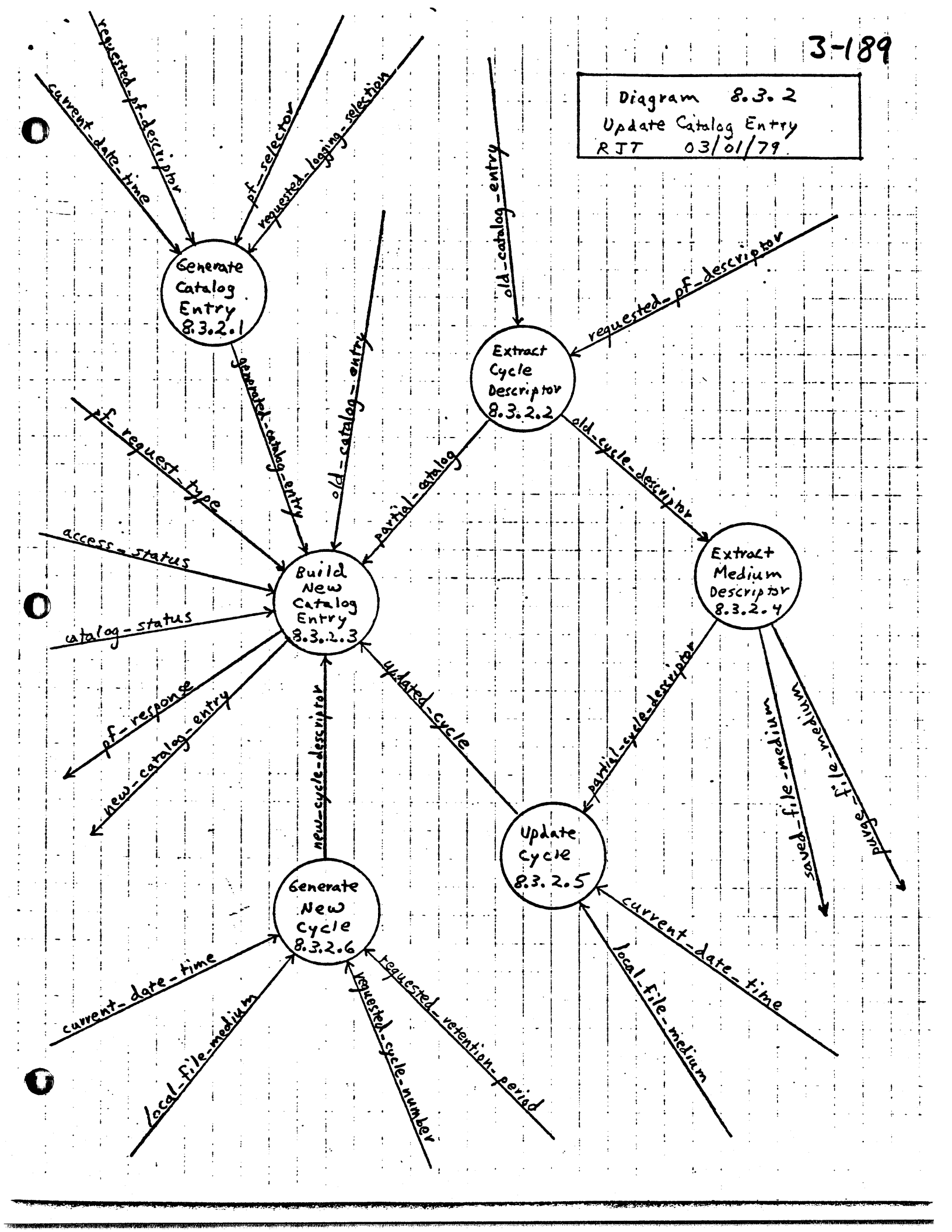


Diagram 8.3.2
Update Catalog Entry
RJT 03/01/79



```
access_control_info : FLOW
  = allowed_usage_mode + allowed_share_mode;

access_status : FLOW
  = (valid_access ! invalid_password ! invalid_user
    ! invalid_usage);

account_identification : FLOW
  = account_name;

accounting_environment : FLOW
  = family_name + account_name + project_name + user_name;

accounting_family : FLOW
  = family_name;

accounting_user : FLOW
  = user_name;

account_project_request : FLOW
  = account_project_request_code;

account_project_response : FLOW
  = account_identification + project_identification;

allowed_share_mode : FLOW
  = share_mode;

allowed_usage_mode : FLOW
  = usage_mode;

alternate_access_control : FLOW
  = alternate_access_family + alternate_access_account
   + alternate_access_project + alternate_access_user
   + alternate_access_usage_mode + alternate_access_share_mode
   + alternate_access_statistics;

alternate_access_statistics : FLOW
  = alternate_access_date_time + alternate_access_count
   + alternate_access_last_cycle;
```

```
build_new_catalog_entry : PROCESS 8.3.2.3. =
  CASE pf_request_type OF
  =attach,get,purge=
    IF catalog_status = missing_catalog THEN
      file_status := missing_file.
    ELSE
      IF cycle_status = missing_cycle THEN
        file_status := missing_cycle.
      ELSE
        file_status := good_status.
        IF pf_request_type = purge THEN
          Send new_catalog_entry formed form partial_catalog.
        ELSE
          Send new_catalog_entry formed from old_catalog_entry.
        IFEND.
      IFEND.
    IFEND.
  =define,save,replace=
    IF catalog_status = missing_catalog THEN
      file_status := good_status.
      Send new_catalog_entry formed from generated_catalog_entry
        and new_cycle_descriptor.
    ELSE
      IF cycle_status = missing_cycle THEN
        file_status := good_status.
        Send new_catalog_entry formed by appending
          new_cycle_descriptor to old_catalog_entry.
      ELSE
        IF pf_request_type = replace THEN
          file_status := good_status.
          Send new_catalog_entry formed from partial_catalog
            and updated_cycle.
        ELSE
          file_status := duplicate_cycle.
        IFEND.
      IFEND.
    IFEND.
  CASEND.
PROCESS_END:


build_pf_selector : PROCESS 8.5. =
  IF requested_pf_name = NULL THEN
    selected_pf_name := requested_local_file.
  ELSE
    selected_pf_name := requested_pf_name.
  IFEND.
  IF requested_family = NULL THEN
    selected_family := requester_family.
  ELSE
    selected_family := requested_family.
  IFEND.
  IF requested_user = NULL THEN
```

```
      selected_user := requester_user.
    ELSE
      selected_user := requested_user.
    IFEND.
    IF requested_catalog = NULL THEN
      selected_catalog := default_catalog.
    ELSE
      selected_catalog := requested_catalog.
    IFEND.
    IF requested_set = NULL THEN
      selected_set := default_set.
    ELSE
      selected_set := requested_set.
    IFEND.
  PROCESS_END;

  build_requester_identification : PROCESS 8.6. =
    Issue user_identification_request.
    requester_family := family_identification.
    requester_user := user_identification.
    Issue account_project_request.
    requester_account := account_identification.
    requester_project := project_identification.
  PROCESS_END;

  catalog_entry : FLOW
    = owner_identification + pf_name + password
     + logging_selection + owner_access_control
     + [<alternate_access_control>] + <cycle_descriptor>;

  catalog_index : FLOW
    = {some key to allow random access to a catalog entry}
    ;

  catalog_result : FLOW
    = catalog_status + old_catalog_entry;

  catalog_set_request : FLOW
    = catalog_set_request_code;

  catalog_set_response : FLOW
    = default_catalog + default_set;

  catalog_status : FLOW
    = (good_catalog | missing_catalog);
```

```
change_descriptor : FLOW
  = new_pf_name + new_cycle_number + new_password;

copy_file_pair : FLOW
  = source_file + destination_file;

creation_status : FLOW
  = (good_cycle | duplicate_cycle | missing_cycle);

current_date_time : FLOW
  = date + time;

cycle_allocation_descriptor : FLOW
  = file_medium_descriptor;

cycle_attach_status : FLOW
  = read_count + append_count + modify_count + execute_count
   + write_count + exclusive_use_flag;

cycle_descriptor : FLOW
  = cycle_number + cycle_statistics + cycle_retention_period
   + cycle_allocation_descriptor + cycle_attach_status;

cycle_statistics : FLOW
  = cycle_creation_date_time
   + cycle_modification_date_time + cycle_access_date_time
   + cycle_access_count;

cycle_status : FLOW
  = (good_cycle | missing_cycle | busy_cycle);

date_time_request : FLOW
  = program_request;

date_time_response : FLOW
  = program_response;

default_catalog : FLOW
  = catalog_name;
```

```
default_set : FLOW
  = set_name;

destination_file : FLOW
  = local_file_name;

determine_access_status : PROCESS 8.3.1.2. =
  IF (usage_status = valid_usage) AND (password_status =
    valid_password) THEN
    access_status := valid_access.
  ELSE
    IF usage_status = invalid_user THEN
      access_status := invalid_user.
    ELSE
      IF password_status = invalid_password THEN
        access_status := invalid_password.
      ELSE
        access_status := invalid_usage.
      IFEND.
    IFEND.
  IFEND.
PROCESS_END;

device_request : FLOW
  = local_file_name + [set_name];

environment : FLOW
  = accounting_environment + date + time + default_set
   + default_catalog;

establish_file_access : PROCESS 8.3.4. =
  Issue return_request_response with requested_local_file.
  CASE of_request_type OF
  =get=
    Issue device_request_response with local_file_name.
    Put file_medium_descriptor in file_medium_table using
      saved_file_medium and temporary_file_name.
    Issue copy_file_pair using temporary_file_name as the
      source file and requested_local_file as the destination
      file.
    Issue return_request_response with temporary_file_name.
  =attach=
    Put file_medium_descriptor in file_medium_table using
      saved_file_medium and requested_local_file.
  =ELSE
  CASEND.
PROCESS_END;
```

```
evict_file : PROCESS 8.3.5. =
  IF ((pf_request_type = replace) OR (pf_request_type = purge))
    THEN
    Issue FAT_deletion_request_response with FAT.
  IFEND.
PROCESS_END:

extract_access_control_info : PROCESS 8.3.1.4. =
  IF (requester_user = owner_user) AND (requester_family =
    owner_family) THEN
    allowed_usage_mode := owner_usage_mode.
    allowed_share_mode := owner_share_mode.
  ELSE
    FOR each alternate_access_control OF old_catalog_entry
      DO UNTIL MATCH
      {NULL fields of alternate_access_control match anything}
      MATCH :=
        (alternate_access_family = requester_family) AND
        (alternate_access_account = requester_account) AND
        (alternate_access_project = requester_project) AND
        (alternate_access_user = requester_user).
    FOREND.
    IF MATCH THEN
      allowed_usage_mode := alternate_access_usage_mode OF
        SELECTED alternate_access_control.
      allowed_share_mode := alternate_access_share_mode OF
        SELECTED alternate_access_control.
    ELSE
      access_control_info := NULL.
    IFEND.
  IFEND.
PROCESS_END:

extract_cycle_descriptor : PROCESS 8.3.2.2. =
  FOR each cycle_descriptor OF old_catalog_entry DO
    IF requested_cycle = cycle_number THEN
      old_cycle_descriptor := SELECTED cycle_descriptor.
      partial_catalog := old_catalog_entry - SELECTED
        cycle_descriptor.
    IFEND.
  FOREND.
PROCESS_END:

extract_medium_descriptor : PROCESS 8.3.2.4. =
  saved_file_medium := cycle_allocation_descriptor OF
    old_cycle_descriptor.
  purge_file_medium := cycle_allocation_descriptor OF
    old_cycle_descriptor.
  partial_cycle_descriptor := old_cycle_descriptor -
```

```
      cycle_allocation_descriptor OF old_cycle_descriptor.
PROCESS_END;

family_identification : FLOW
  = family_name;

fat_deletion_request : FLOW
  = fat_deletion_request_code + purge_fat
    {drop space described by the purge_fat};

file_info_request : FLOW
  = file_info_request_code + local_file_name;

file_info_response : FLOW
  = file_info_status + file_info_set;

file_info_set : FLOW
  = set_name;

file_info_status : FLOW
  = (found | not_found);

file_status : FLOW
  = (good_status | missing_file | missing_cycle
    | duplicate_cycle);

generate_catalog_entry : PROCESS 8.3.2.1. =
  generated_family := selected_family.
  generated_set := selected_set.
  generated_catalog_name := selected_catalog.
  generated_user := selected_user.
  generated_pf_name := selected_pf_name.
  generated_password := requested_password.
  generated_logging_selection := requested_logging_selection.
  generated_usage_mode := write.
  generated_share_mode := write.
  generated_access_date_time := current_date_time.
  generated_access_count := 0.
  generated_last_cycle accessed := 0.
PROCESS_END;

generated_access_count : FLOW
  = access_count;
```

```
generated_access_date_time : FLOW
  = owner_access_date_time;

generated_catalog_entry : FLOW
  = generated_family + generated_set
   + generated_catalog_name + generated_user
   + generated_pf_name + generated_password
   + generated_logging_selection + generated_usage_mode
   + generated_share_mode + generated_last_access_date
   + generated_access_date_time + generated_last_cycle_accessed;

generated_catalog_name : FLOW
  = catalog_name;

generated_family : FLOW
  = family_name;

generated_last_access_time : FLOW
  = time;

generated_last_cycle_accessed : FLOW
  = cycle_number;

generated_logging_selection : FLOW
  = logging_selection;

generated_password : FLOW
  = password;

generated_pf_name : FLOW
  = pf_name;

generated_set : FLOW
  = set_name;

generated_share_mode : FLOW
  = share_mode;

generated_usage_mode : FLOW
  = usage_mode;
```

```
    generated_user : FLCW
      = user_name:

    generate_file_status : PROCESS 8.3.2.3.1. =
      {This process is not used.}
      CASE pf_request_type OF
      =attach,get,purge=
        IF catalog_status = missing_catalog THEN
          file_status := missing_file.
        ELSE
          IF cycle_status = missing_cycle THEN
            file_status := missing_cycle.
          ELSE
            file_status := good_status.
          IFEND.
        IFEND.
      =define,save=
        IF ((catalog_status = good_catalog) AND (cycle_status =
          good_cycle)) THEN
          file_status := duplicate_cycle.
        ELSE
          file_status := good_status.
        IFEND.
      =replace=
        file_status := good_status.
      CASEND.
    PROCESS_END:

    generate_new_cycle : PROCESS 8.3.2.6. =
      WITH new_cycle_descriptor DO
        cycle_number := requested_cycle_number.
        cycle_creation_date_time := current_date_time.
        cycle_modification_date_time := current_date_time.
        cycle_access_date_time := current_date_time.
        cycle_access_count := 0.
        cycle_retention_period := requested_retention_period.
        cycle_allocation_descriptor := local_file_medium.
        WITH cycle_attach_status DO
          read_count := 0.
          append_count := 0.
          modify_count := 0.
          execute_count := 0.
          write_count := 0.
          exclusive_use_flag := false.
        WITHEND.
      WITHEND.
    PROCESS_END:

    local_file_medium : FLCW
```

```
    = file_medium_descriptor;

logging_selection : FLOW
    = (log | nolog);

new_catalog_entry : FLOW
    = catalog_entry;

new_cycle_descriptor : FLOW
    = cycle_descriptor;

new_cycle_number : FLOW
    = cycle_number;

new_password : FLOW
    = password;

new_pf_name : FLOW
    = pf_name;

obtain_local_file : PROCESS 8.3.6. =
  CASE pf_request_type OF
  =save,replace=
    Issue device_request with temporary_file_name and
      selected_set of pf_selector.
    Issue copy_file_pair using requested_local_file as
      source_file and temporary_file_name as destination_file.
    Extract file_medium_descriptor for temporary_file_name
      from file_medium_table, sending it as local_file_medium.
    Issue return_request using temporary_file_name.
  =define=
    Issue file_info_request using requested_local_file.
    IF file_info_status = not_found THEN
      Issue device_request with requested_local_file and
        selected_set of pf_selector.
      Extract file_medium_descriptor for requested_local_file
        from file_medium_table, sending it as local_file_medium.
    ELSE
      IF file_info_set = selected_set of pf_selector THEN
        Extract file_medium_descriptor for requested_local_file
          from file_medium_table, sending it as
          local_file_medium.
      ELSE
        Issue rename_request with new_name set to
          temporary_file_name and old_name set to
          requested_local_file.
```

```
            Issue copy_file_pair using temporary_file_name as
              source_file and requested_local_file as
              destination_file.
            Extract file_medium_descriptor for requested_local_file
              from file_medium_table, sending it as
              local_file_medium.
          IFEND.
        IFEND.
      ELSE
      CASEND.
    PROCESS_END:

    old_catalog_entry : FLOW
      = catalog_entry;

    old_cycle_descriptor : FLOW
      = cycle_descriptor;

    owner_access_control : FLOW
      = owner_usage_mode + owner_share_mode
       + owner_access_statistics;

    owner_access_statistics : FLOW
      = owner_access_date_time + owner_access_count
       + owner_access_last_cycle;

    owner_catalog : FLOW
      = catalog_name;

    owner_family : FLOW
      = family_name;

    owner_identification : FLOW
      = owner_family + owner_set + owner_catalog + owner_user;

    owner_set : FLOW
      = set_name;

    owner_user : FLOW
      = user_name;

    partial_cycle_descriptor : FLOW
      = cycle_descriptor - cycle_allocation_descriptor;
```

```
password_status : FLCW
  = (valid_password | invalid_password);

permit_account : FLCW
  = account_name;

permit_descriptor : FLCW
  = permit_family + permit_account + permit_project + permit_user;

permit_family : FLCW
  = family_name;

permit_project : FLCW
  = project_name;

permit_user : FLCW
  = user_name;

pf_descriptor : FLCW
  = pf_name + cycle_number + password + family_name
   + user_name + catalog_name + set_name;

pf_properties : FLCW
  = logging_selector + retention_period + size;

pf_request : FLCW
  = pf_request_type + requested_pf_descriptor
   + requested_usage_descriptor + requested_logging_selection
   + requested_retention_period + requested_size
   + change_descriptor + permit_descriptor
   + requested_local_file;

pf_request_type : FLCW
  = (define | attach | get | save | replace | purge | change
   | permit);

pf_selector : FLCW
  = selected_pf_name + selected_family + selected_user
   + selected_catalog + selected_set;
```

```
project_identification : FLOW
  = project_name;

purge_file_medium : FLOW
  = file_medium_descriptor;

rename_request : FLOW
  = rename_request_code + old_name + new_name;

requested_catalog_name : FLOW
  = catalog_name;

requested_cycle_number : FLOW
  = cycle_number;

requested_family : FLOW
  = family_name;

requested_local_file : FLOW
  = local_file_name;

requested_logging_selection : FLOW
  = logging_selection;

requested_password : FLOW
  = password;

requested_pf_descriptor : FLOW
  = requested_pf_name + requested_cycle_number
    + requested_password + requested_family + requested_user
    + requested_catalog + requested_set;

requested_pf_name : FLOW
  = pf_name;

requested_retention_period : FLOW
  = retention_period;

requested_set : FLOW
  = set_name;
```

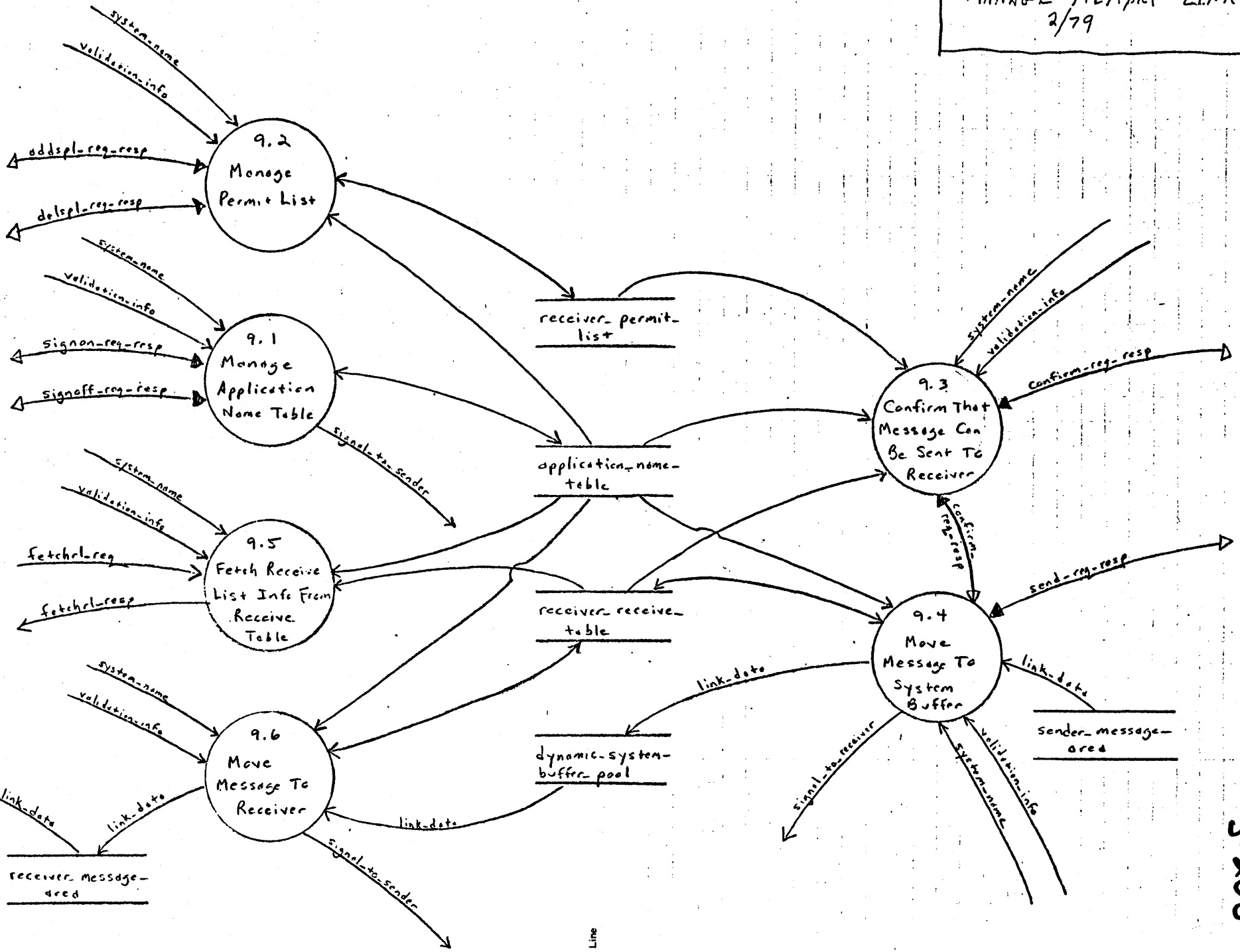
```
requested_share_mode : FLOW
  = share_mode;

requested_usage_descriptor : FLOW
  = requested_usage_mode + requested_share_mode
   + requested_wait_selection;

requested_usage_mode : FLOW
  = usage_mode;

requested_user : FLOW
  = user_name;

requested_wait_selection : FLOW
  = wait_selection;

requester_account : FLOW
  = account_name;

requester_family : FLOW
  = family_name;

requester_identification : FLOW
  = requester_family + requester_user + requester_account
   + requester_project;

requester_project : FLOW
  = project_name;

requester_user : FLOW
  = user_name;

return_request : FLOW
  = return_request_code + local_file_name;

saved_file_medium : FLOW
  = file_medium_descriptor;

share_mode : FLOW
  = (exclusive | read | append | modify | execute | write);
```

```
source_file : FLOW
  = local_file_name;

update_cycle : PROCESS 8.3.2.5. =
  Merge local_file_medium and partial_cycle_descriptor
    forming updated_cycle.
PROCESS_END;

updated_catalog_entry : FLOW
  = catalog_entry;

usage_descriptor : FLOW
  = usage_mode + share_mode + wait_selection;

usage_mode : FLOW
  = (read ! write ! append ! modify ! execute);

usage_status : FLOW
  = (valid_usage ! invalid_user ! invalid_usage);

user_identification : FLOW
  = user_name;

user_identification_request : FLOW
  = user_identification_request_code;

user_identification_response : FLOW
  = family_identification + user_identification;

validate_password : PROCESS 8.3.1.1. =
  IF requested_password = password field of
    old_catalog_entry THEN
    password_status := valid_password.
  ELSE
    password_status := invalid_password.
  IFEND.
PROCESS_END;

validate_usage : PROCESS 8.3.1.3. =
  IF access_control_info = NULL THEN
    usage_status := invalid_user.
```

```
      ELSE
        IF (requested_usage_mode = allowed_usage_mode) AND
          (requested_share_mode = allowed_share_mode) THEN
          usage_status := valid_usage.
        ELSE
          usage_status := invalid_usage.
        IFEND.
      IFEND.
    PROCESS_END;


  wait_selection : FLOW
    = (wait | nowait);
```

3.9 MANAGE_MEMORY_LINK

9.0
MANAGE MEMORY LINK
2/79



3-208

Diagram ID: Name: Description:

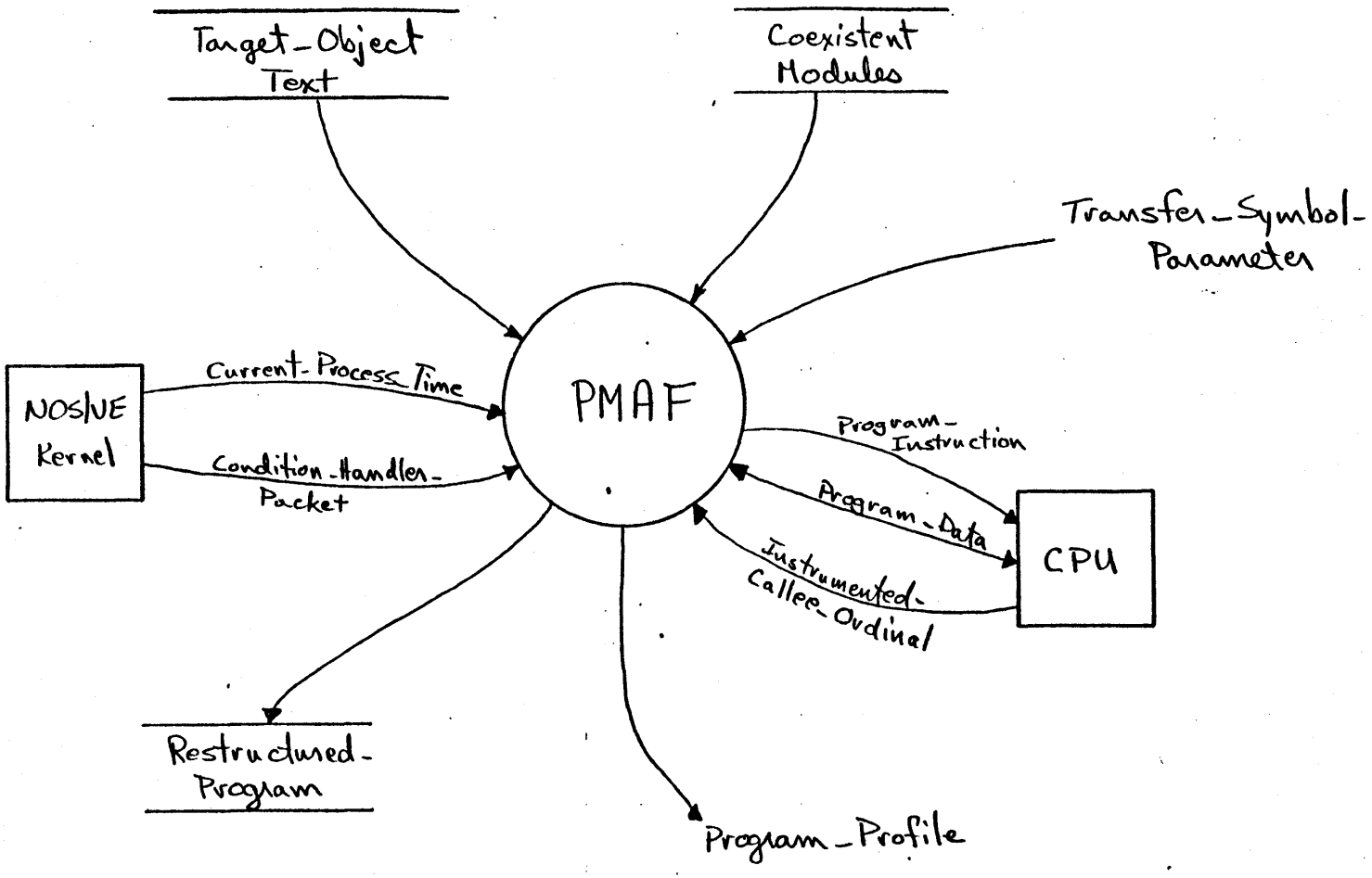
```
addspl_req : FLOW
  = receiver_application_name + sender_application_name;
addspl_req_resp : FLOW
  = (addspl_req | addspl_resp);
addspl_resp : FLOW
  = (normal | (abnormal + (too_many_permits | invalid_receiver_name_format
     | invalid_sender_name_format | permitting_another_application)));
application_name : FLOW
  = (pre_defined_name | system_name);
application_name_table : FILE
  = 0 <application_name
     + receive_table_pointer
     + permit_list_pointer> max_applications;
confirm_req : FLOW
  = sender_application_name + receiver_application_name;
confirm_req_resp : FLOW
  = (confirm_req | confirm_resp);
confirm_resp : FLOW
  = (normal | (abnormal + (confirming_for_another_application
     | sender_not_permitted | receiver_not_signed_on
     | receiver_max_msgs_exceeded | prior_msg_not_received)));
delspl_req : FLOW
  = receiver_application_name + sender_application_name;
delspl_req_resp : FLOW
  = (delspl_req | delspl_resp);
delspl_resp : FLOW
  = (normal | (abnormal + (deleting_for_another_application
     | invalid_receiver_name_format | invalid_sender_name_format)));
dynamic_system_buffer_location : ELEMENT
  = { index, pointer, or something indicating location of one message }
  ;
dynamic_system_buffer_pool : FILE
  = { bunch of wired words to store "in transit" messages in }
  ;
fetchrl_req : FLOW
  = receiver_application_name + [sender_application_name]
    + receive_list_location;
fetchrl_req_resp : FLOW
  = (fetchrl_req | fetchrl_resp);
fetchrl_resp : FLOW
  = ((normal + receive_list) | (abnormal + (fetching_for_another_application
     | invalid_reciever_name_format | invalid_sender_name_format)));
link_data : FLOW
  = <message>;
memory_link_req : FLOW
  = (signon_req | signoff_req | addspl_req | delspl_req | confirm_req
     | send_req | fetchrl_req | receive_req);
memory_link_req_resp : FLOW
  = (memory_link_req | memory_link_resp);
memory_link_resp : FLOW
  = (signon_resp | signoff_resp | addspl_resp | delspl_resp | confirm_resp
     | send_resp | fetchrl_resp | receive_resp);
permit_list : FILE
  = 0 <sender_application_name> max_permits;
permit_list_pointer : ELEMENT
  = { pointer to beginning of thread or fixed_size area for a }
    {   particular receiver application                       }
  ;
```

```
pre_defined_name : ELEMENT
  = 1 <ascii> osc$name_size { each pre_defined application has at least
                            { one unique pre-defined name };
privileged_application : FLOW
  = (170_system_application ! job_class { 180 privileged task } );
receiver_application_name : FLOW
  = application_name;
receiver_message_area : FILE
  = { receiver-owned buffer }
  ;
receiver_permit_list : FILE
  = 0 <permit_list> max_applications;
receiver_receive_table : FLOW
  = 0 <receive_table> max_applications;
receive_list : FLOW
  = 0 <sender_application_name
      + sender_arbitrary_info
      + receive_table_index
      + message_length> max_messages;
receive_message_area : FILE
  = { receiver-owned buffer }
  ;
receive_req : FLOW
  = receiver_application_name + receive_table_index + receive_buffer_location
    + receive_buffer_length + receive_signal_option;
receive_req_resp : FLOW
  = (receive_req ! receive_resp);
receive_resp : FLOW
  = ((normal + message_length + sender_arbitrary_info)
     ! (abnormal + (receiving_for_another_application
     ! invalid_receive_table_index ! message_too_large
     ! invalid_buffer_location ! invalid_signal_option)));
receive_signal_option : FLOW
  = (signal_sender ! dont_signal_sender);
receive_table : FILE
  = 0 <sender_application_name
      + sender_arbitrary_info
      + dynamic_system_buffer_location
      + message_length
      + send_signal_option
      + receive_signal_option> max_messages;
receive_table_index : FLOW
  = { specifies exactly which message this is }
  ;
receive_table_pointer : ELEMENT
  = { pointer to beginning of thread or fixed-size area for a }
    {   particular receiver application                        }
  ;
sender_application_name : FLOW
  = application_name;
sender_arbitrary_info : ELEMENT
  = 0..256 { upper bound is really tbd };
sender_message_area : FILE
  = { sender-owned buffer }
  ;
send_req : FLOW
  = sender_application_name + receiver_application_name
    + sender_arbitrary_info + message_buffer_location + message_length
```

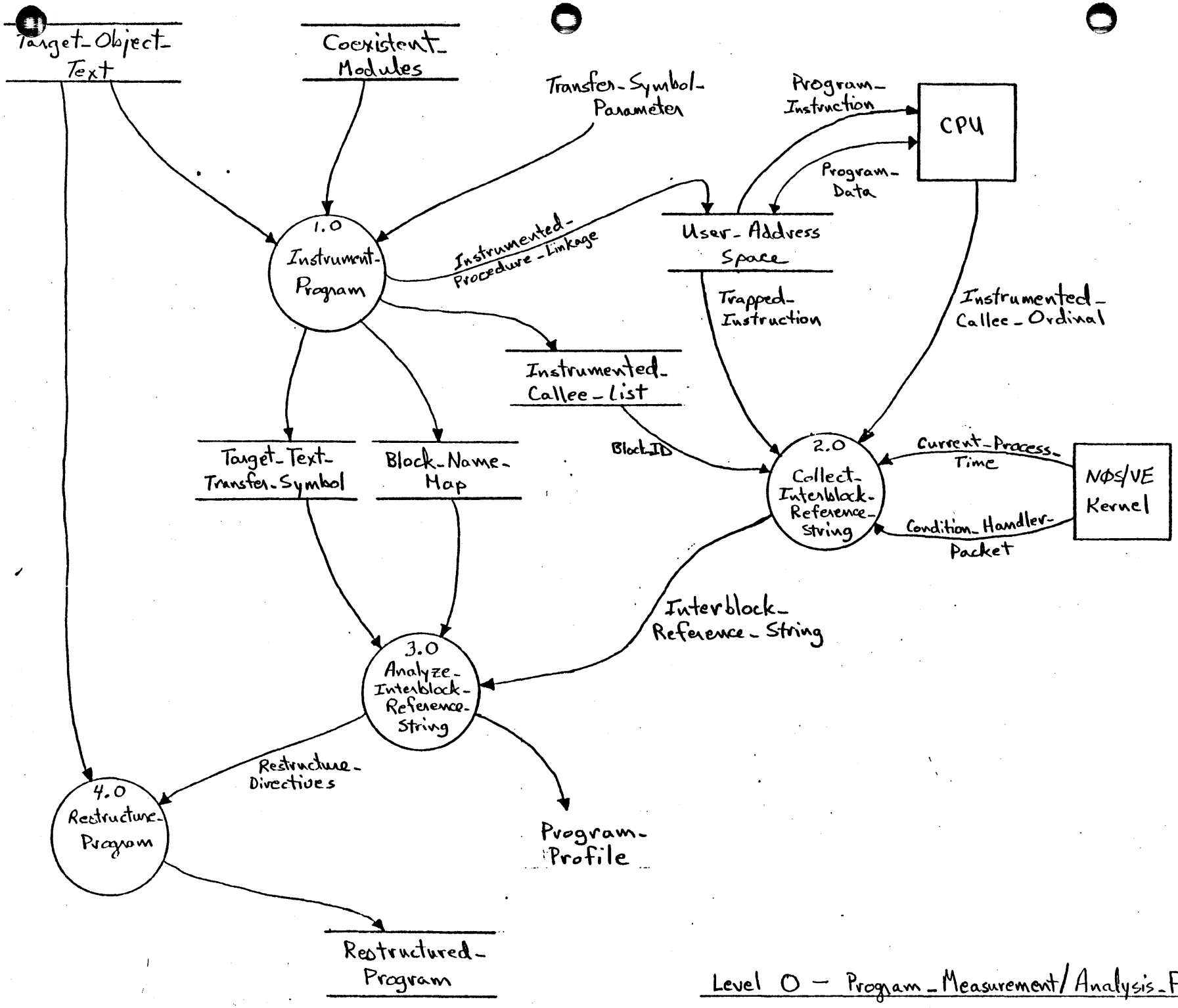
MEMLINK 79/02/26. 14.21.35.

```
    + send_signal_option;
send_req_resp : FLOW
  = (send_req ! send_resp);
send_resp : FLOW
  = (normal ! (abnormal + (sending_for_another_application
     ! sender_not_permitted ! receiver_not_signed_on
     ! receiver_max_msgs_exceeded ! prior_msg_not_received)));
send_signal_option : FLOW
  = (signal_receiver ! dont_signal_receiver);
signal_to_receiver : FLOW
  = {"there is a message waiting for you"}
  ;
signal_to_sender : FLOW
  = {"the message you sent has been received"}
  ;
signoff_req : FLOW
  = receiver_application_name;
signoff_req_resp : FLOW
  = (signoff_req ! signoff_resp);
signoff_resp : FLOW
  = (normal ! (abnormal + (signoff_for_another_application
     ! invalid_application_name)));
signon_req : FLOW
  = [receiver_application_name] + [max_messages];
signon_req_resp : FLOW
  = (signon_req ! signon_resp);
signon_resp : FLOW
  = (normal ! (abnormal + (not_permitted_predefined_name
     ! not_permitted_max_messages ! invalid_receiver_name_format
     ! max_messages_too_large)));
system_name : FLOW
  = (real_state_job_id ! global_task_id);
validated_to_use_memory_link : FLOW
  = (170_user_can_use_017 ! 180_user_can_call_memory_link);
validation_info : FLOW
  = (validated_to_use_memory_link + privileged_application);
```

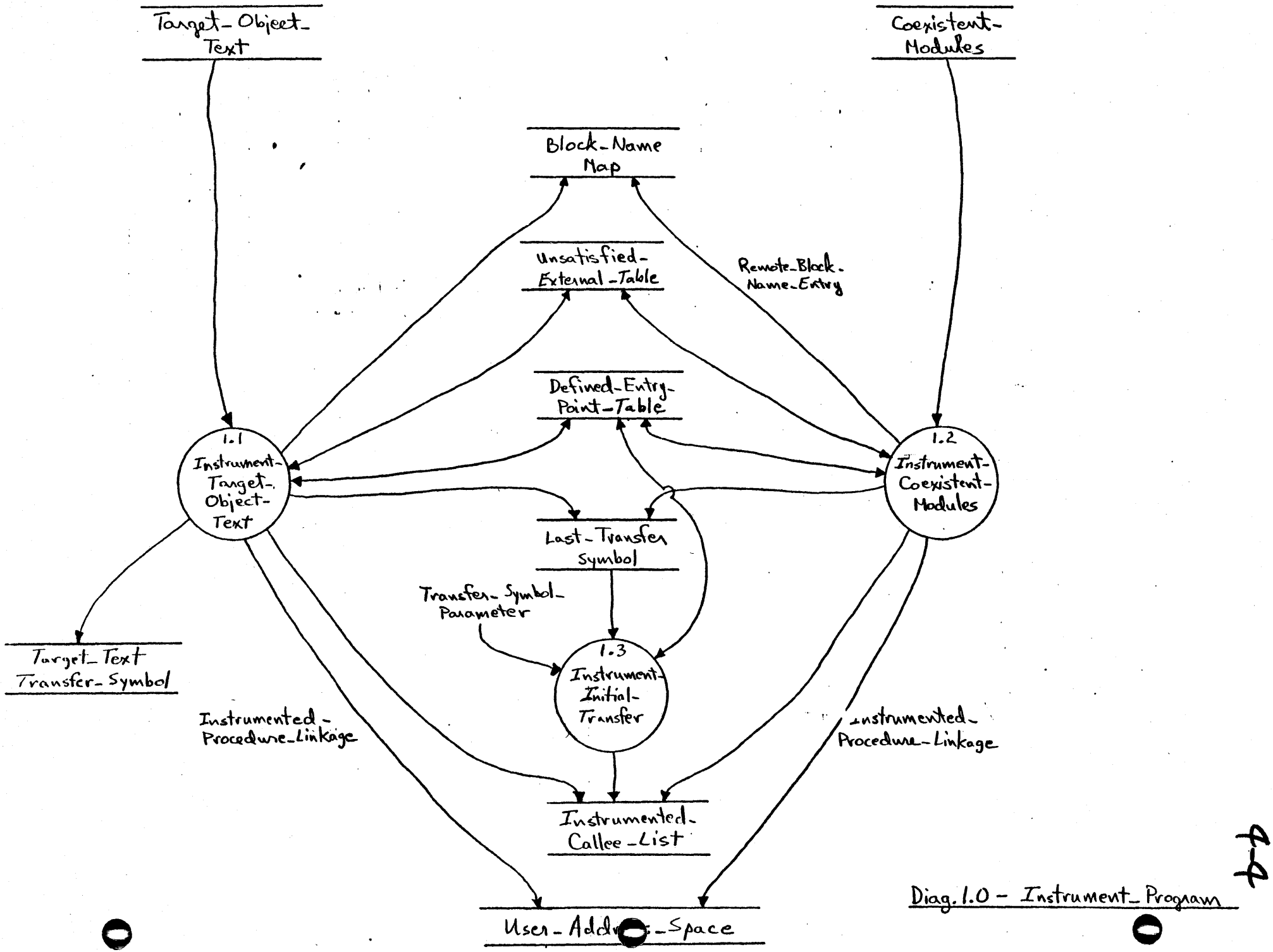
4.0 ANALYZE PROGRAM EXECUTION



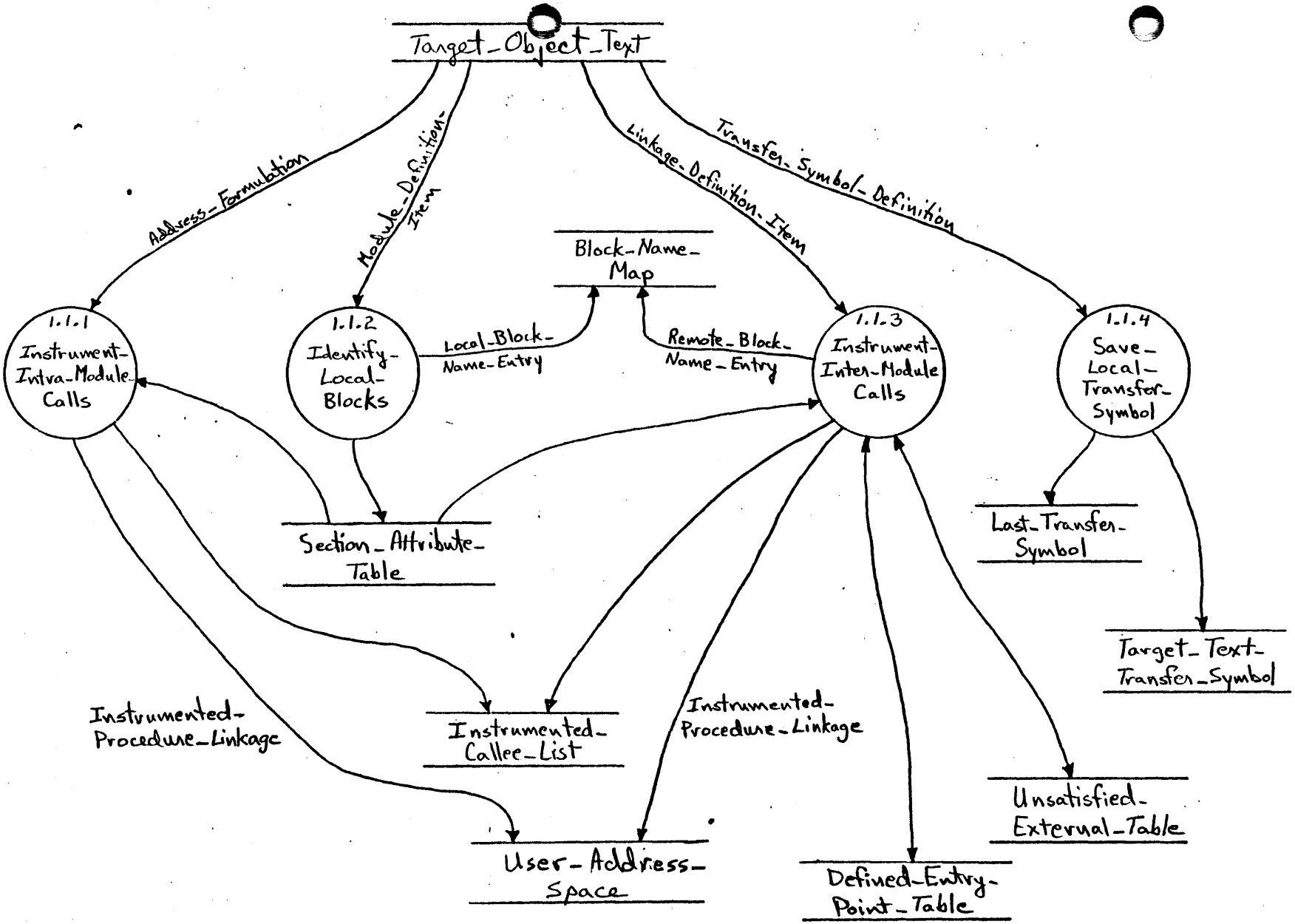
Context Diagram



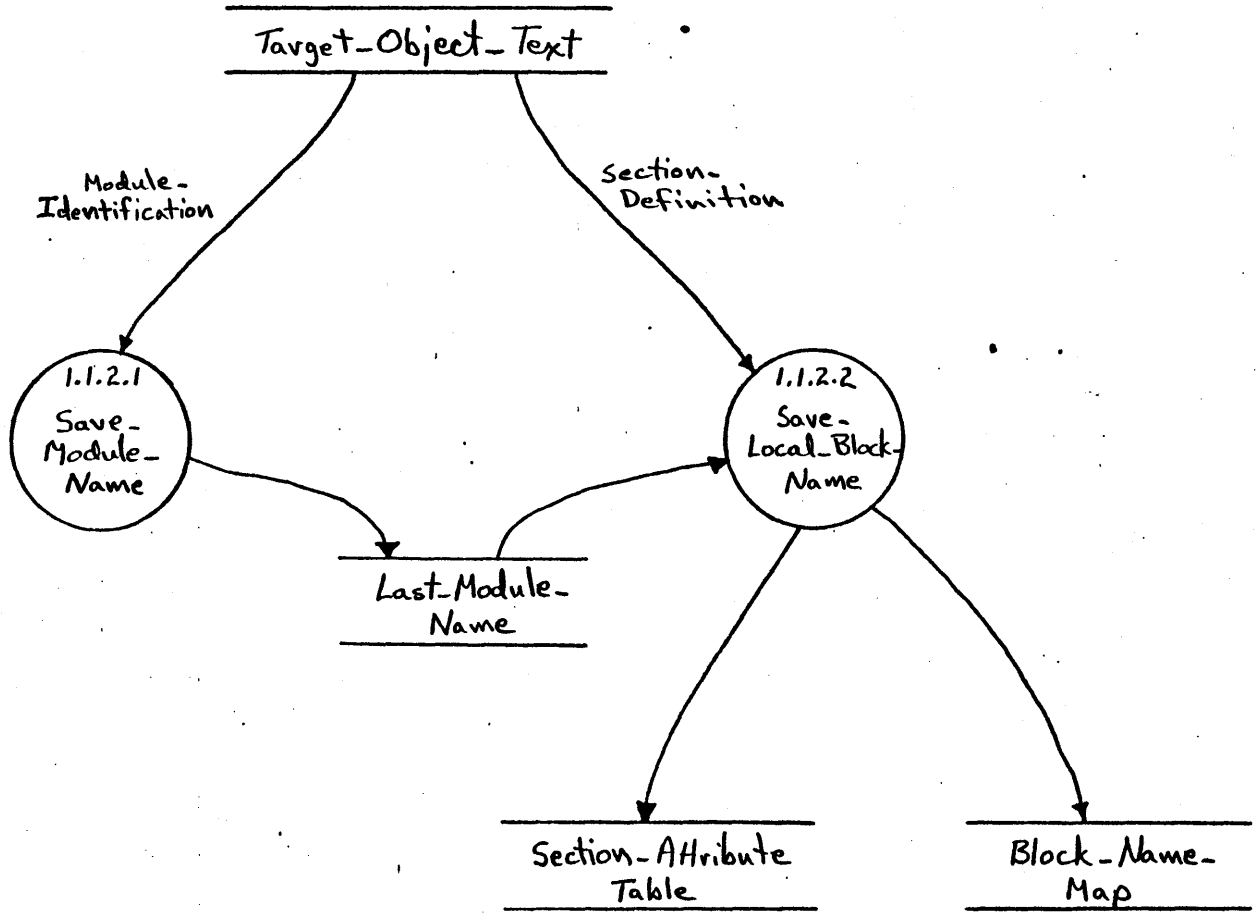
Level 0 — Program_Measurement/Analysis_Facility



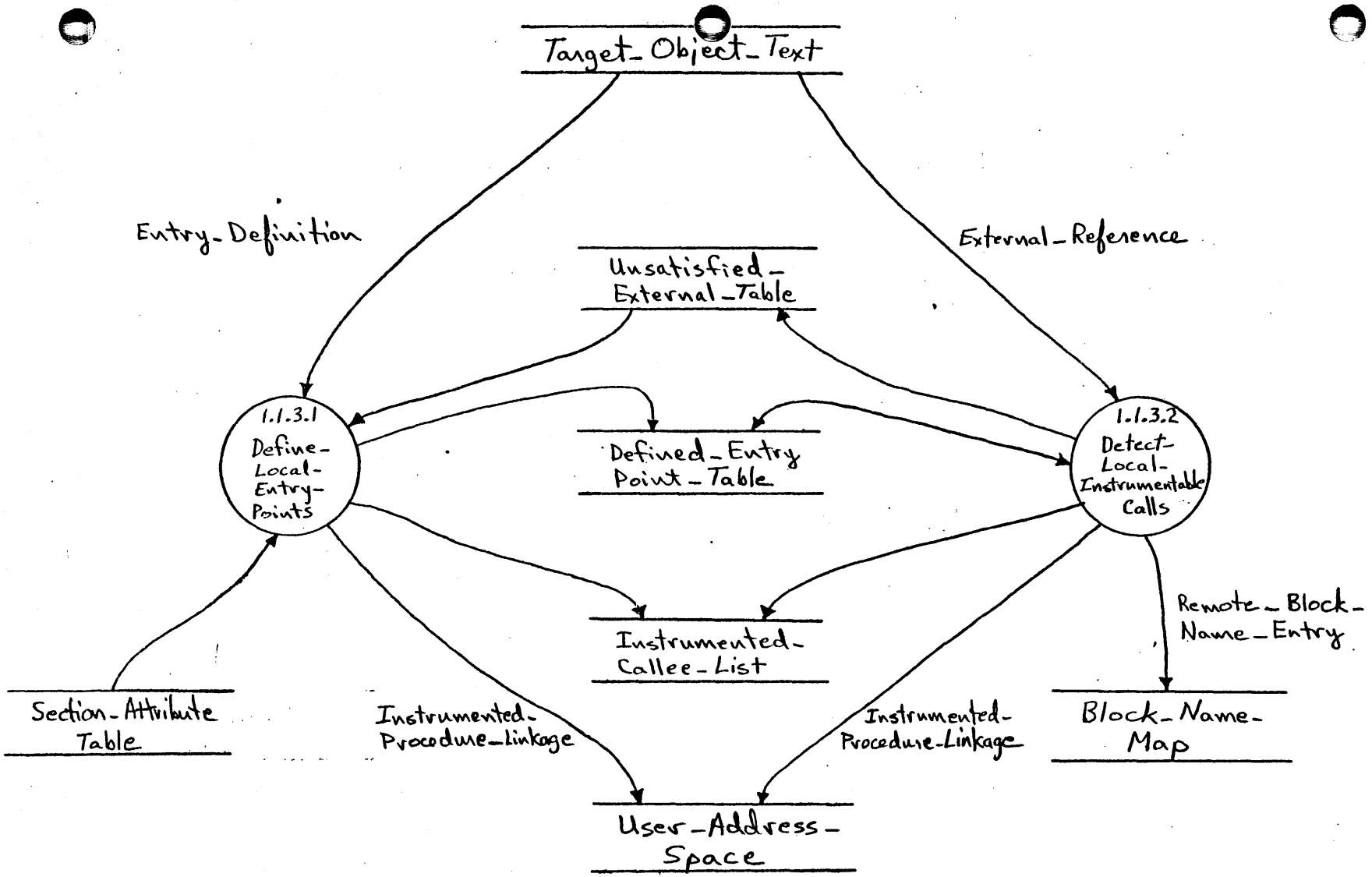
Diag. 1.0 – Instrument_Program



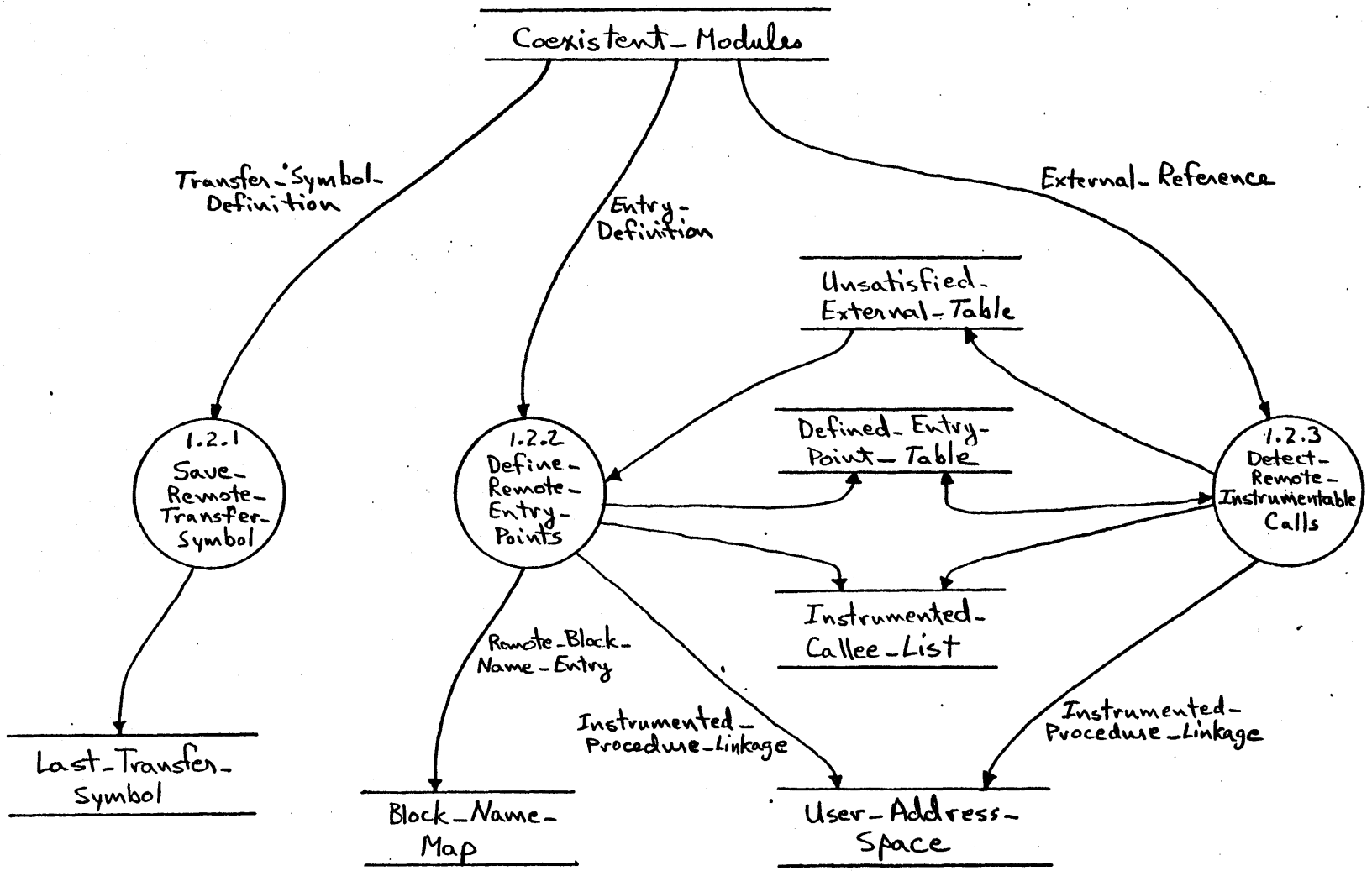
Diag. 1.1 - Instrument_Target_Object_Text



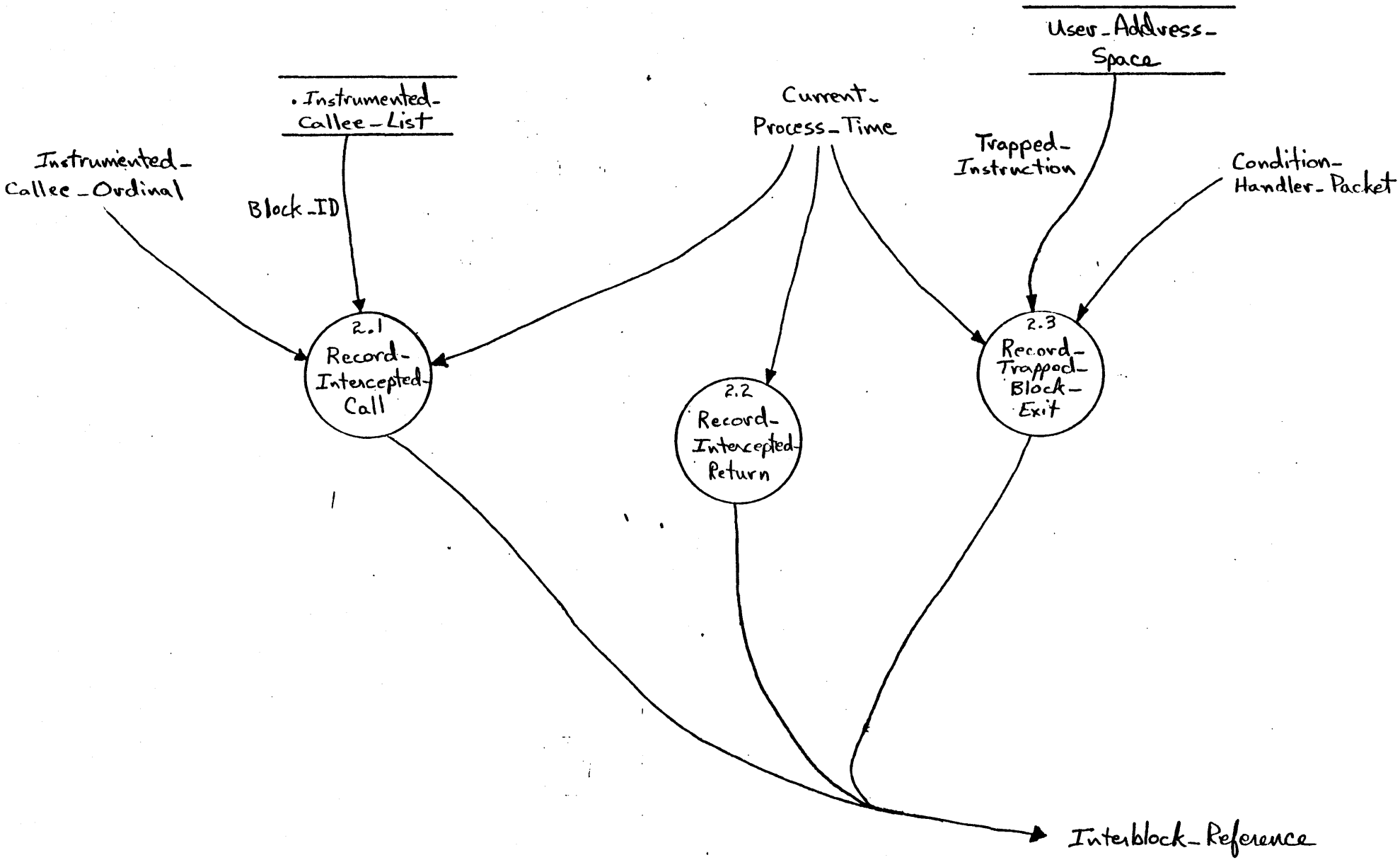
Diag. 1.1.2 - Identify_Local_Blocks



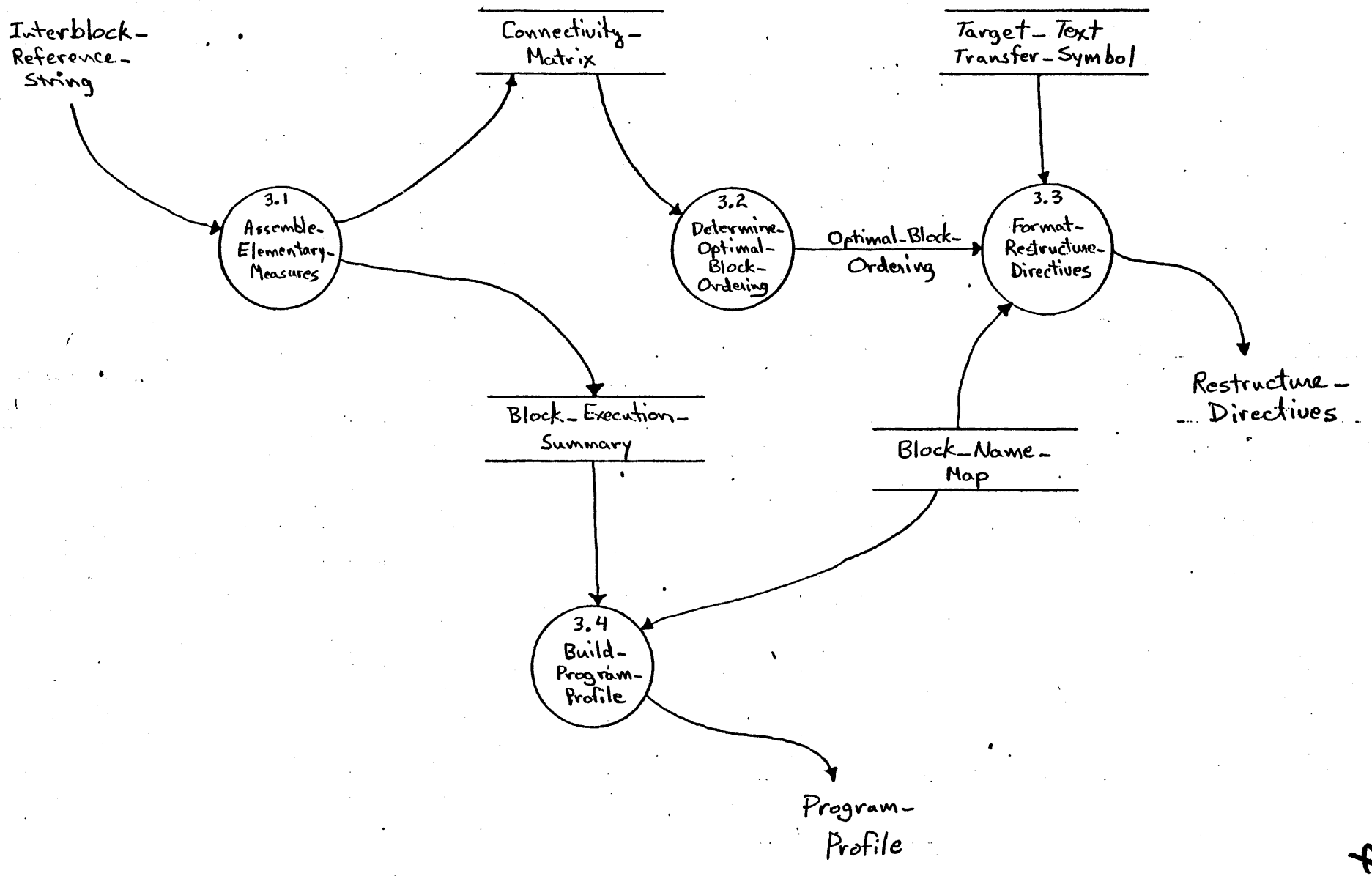
Diag. 1.1.3 - Instrument_Inter_Module_Calls



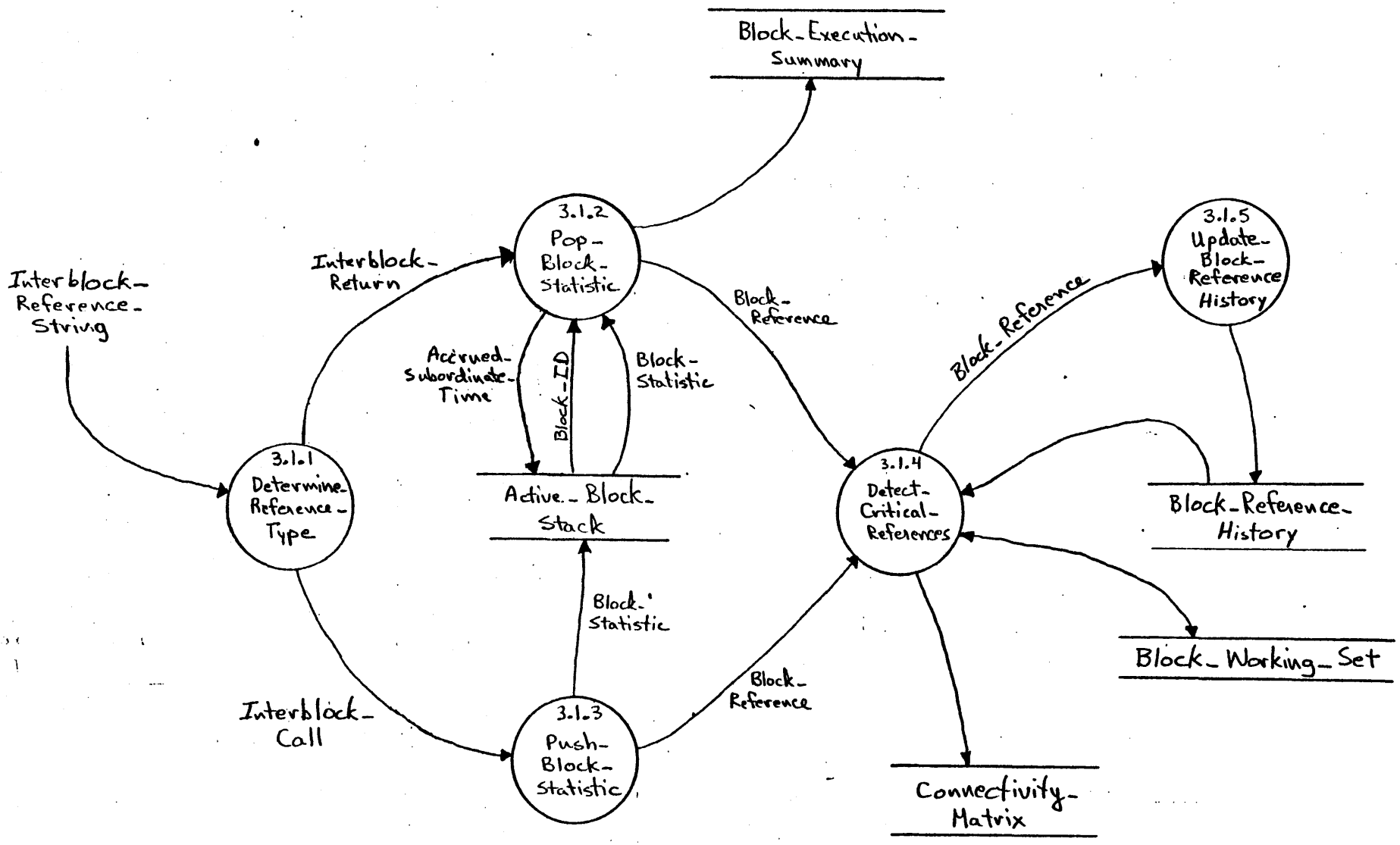
Diag 1.2 - Instrument_Coexistent_Modules



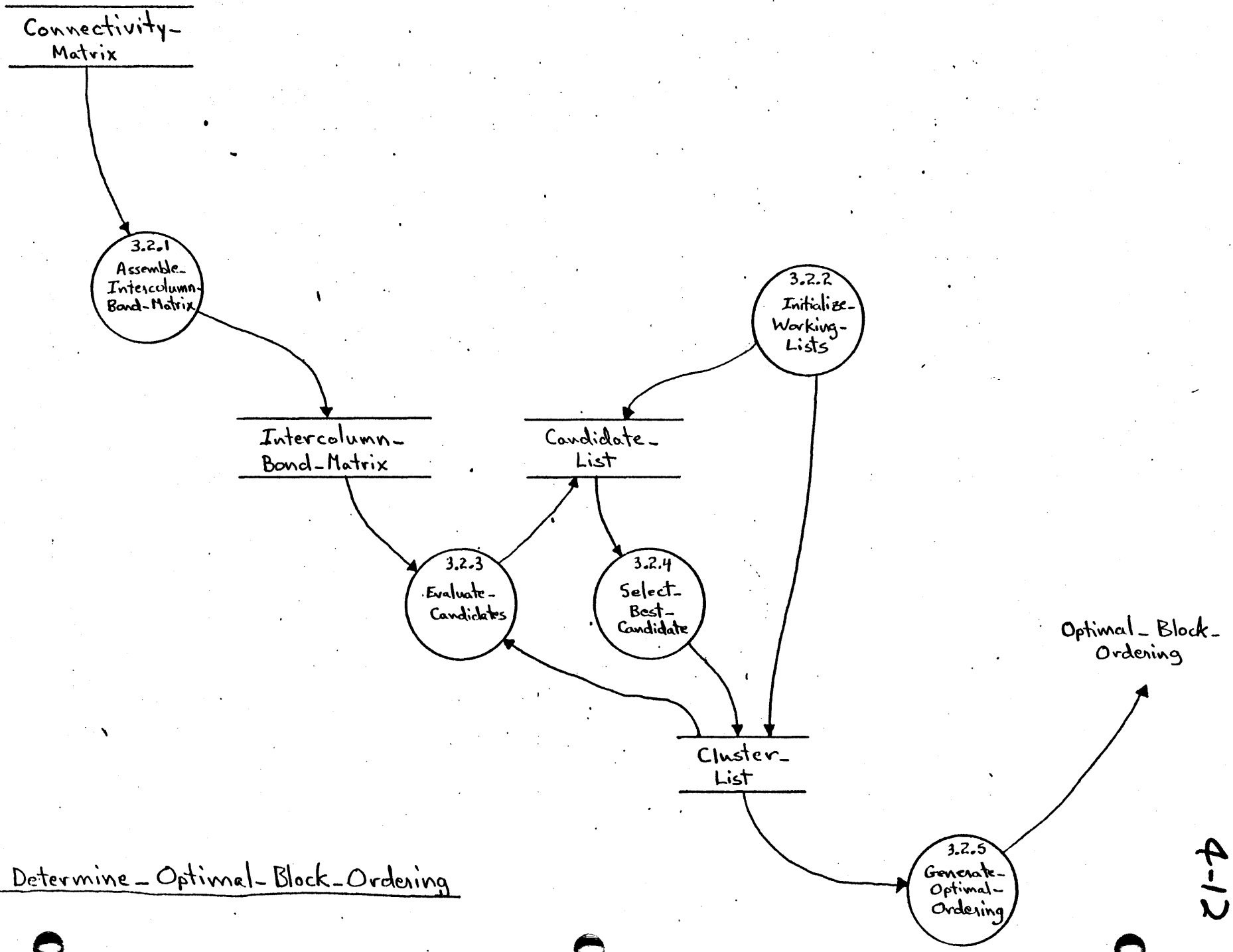
Diag. 2.0 – Collect_Interblock_Reference_String



3.0 Analyze-Interblock-Reference-String



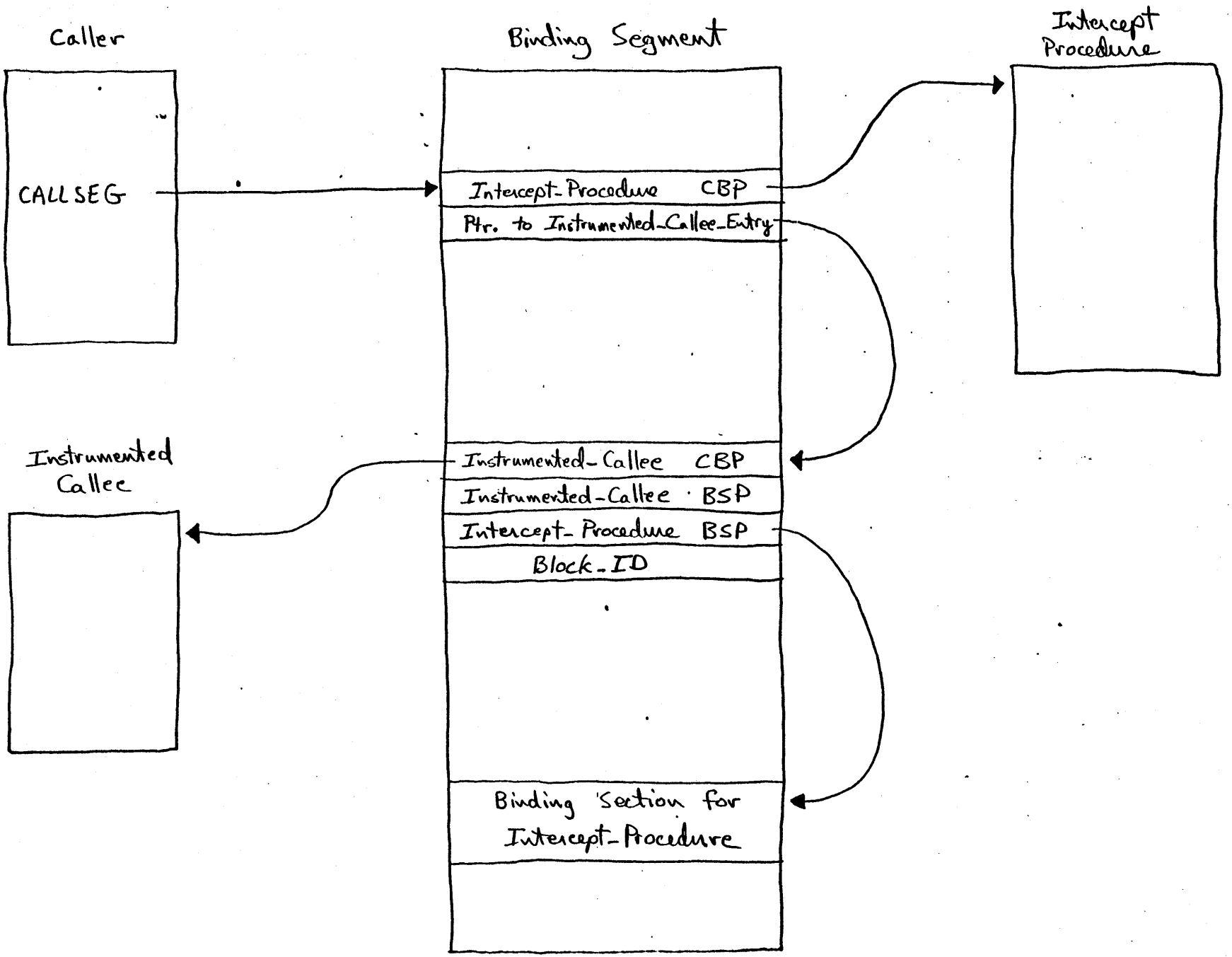
3.1 Assemble_Elementary_Measures



3.2 Determine_Optimal_Block_Ordering

Specification Note

The PMAF structured specification requires that certain procedure linkages in an instrumented program be replaced by a linkage to an intercept procedure plus an index identifying the instrumented callee procedure which is the actual destination of the call. This index is used as a key to access a file which contains the Block_ID of the block containing the procedure and the actual linkage to the instrumented callee. Since the mechanism required to manage the control flows involved in this process do not appear in the structured specification, one might be inclined to believe that some magic is necessary to accomplish such a task. To allay such fears, the following diagram depicts a means of implementing such a transfer control mechanism. In this implementation, the Instrumented_Callee_List is packaged as a binding section of the instrumented program.



Scenario for Possible Implementation of Instrumentation

```
Active_Block_Stack : FILE
   {A LIFO list which is used to keep track of the Block_ID of the currently
   {executing procedure.  Also used to accumulate the execution time accrued
   {by subordinate blocks.
  =    <Block_Statistic>;

Address_Formulation : FLOW
  =    Reference_Type
   +   Reference_Location
   +   Destination_Location;

Best_Position : ELEMENT
   {The ordinal of the current entry in the Cluster_List which the candidate
   {should follow for best clustering affect.  An ordinal of 0 implies the
   {head of the list.
  =;

Block_Chargeback_Percentage : ELEMENT
   {Percentage of total program execution time spent in the associated block,
   {or remote blocks called from the associated block.
  =;

Block_Execution_Summary : FILE
   {The Block_Execution_Summary entry for each Block_Id is used to accumulate
   {the execution time accrued by the associated block.
  =    <Block_Execution_Totals>;

Block_Execution_Totals : FLOW
       *Block_ID
   +   Block_Total
   +   Remote_Total;

Block_ID : FLOW
   {Blocks are the basic units over which measurements are taken and reported.
   {Each block is assigned a unique Block_ID for ease of identification during
   {the analysis phase of PMAF.
  =(   Local_Block_ID
   !   Remote_Block_ID);

Block_Name_Map : FILE
   {For each block, Block_Name_Map defines the
   {symbolic name by which the block is known.
  =    <Block_Name_Map_Entry>;

Block_Name_Map_Entry : FLOW
  =(   Local_Block_Name_Entry
   !   Remote_Block_Name_Entry);

Block_Percentage : ELEMENT
   {Percentage of total program execution time spent in the associated block.
  =;

Block_Reference : FLOW
  =    Local_Block_ID
   +   Reference_Time;

Block_Reference_History : FILE
   {A FIFO list which maintains, in chronological order, all Block_Reference's
```

```
   {occurring in the previous CWS_Interval_Size units of process execution
   {time.
  =     <Block_Reference>;

Block_Statistic : FLOW
  =     Block_ID
   +    Call_Time
   +    Subordinate_Time:

Block_Working_Set : FILE
        {A sequence containing a Local_Block_ID for each block appearing
        {in the current working set.
  =     <Working_Set_Block_ID>:

Candidate_Evaluation : FLOW
  =     Local_Block_ID
   +    Cluster_Merit
   +    Best_Position:

Candidate_List : FILE
   {Identifies those blocks which have not been included in the optimal
   {clustering.
  =     <Candidate_Evaluation>:

Cluster_List : FILE
   {A sequence identifying the relative position of blocks which have been
   {included in the optimal clustering.
  =     <Local_Block_ID>:

Cluster_Merit : ELEMENT
   {An evaluator of the relative merit of including a candidate block in the
   {optimal clustering.
  =:

Coexistent_Modules : FILE
  =     <Object_Text_Module>:

Condition_Handler_Packet : FLOW
   {Data passed to a condition_handler by the NOS/VE trap interrupt
   {processor.
  =     Program_Address
   +    Other:

Connectivity_Matrix : FILE
   {An element of Connectivity_Matrix defines the strength of connection
   {between two blocks.  This value is a local measure of the locality
   {achieved when the associated blocks are clustered next to each other.
   {NOTE...Since this matrix is symmetric, its access must be implemented
   {       such that the order of elements in the indexing
   {       Local_Block_ID_Pair is not relevant.  Thus the matrix will be
   {       realized in some triangular format.
  =<    *Local_Block_ID_Pair
   +    Connection_Strength >:

Current_Process_Time : FLOW
   {The current value for elapsed process execution time.
  =;
```

```
    + + [Reference_Time]
    + [Block_ID];
Interblock_Reference_String : FLOW
  =     <Interblock_Reference>;

Interblock_Reference_Type : ELEMENT
  =(    'call'
    |   'pop'
    |   'return' );

Interblock_Return : FLOW
  =     Reference_Time;

Intercept_Prodecure_Linkage : ELEMENT
    {Linkage to an intercept procedure which will receive control instead
    {of the instrumented callee.  The intercept procedure will be
    {responsible for managing the control transfer to the instrumented
    {callee in an orderly fashion.  This shall include activation of
    {processes to record (in the Interblock_Reference_String) the
    {occurrence of a call to and return from the instrumented callee.
    {The intercept procedure must also take action to provide
    {detection of any attempt by the instrumented callee to POP its
    {way back to previous stack frames.
  =;

Intercolumn_Bond_Matrix : FILE
    {Each element defines the inter-column bond between two columns of the
    {Connectivity_Matrix.  This value is a global measure of the locality
    {achieved when the associated blocks are clustered next to each other.
    {NOTE...This matrix is symmetric and may be represented in triangular
    {format.
  =<    *Local_Block_ID_Pair
    +   Intercolumn_Bond_Strength >;

Last_Module_Name : FILE
    {Always contains the Module_Name from the most recently encountered
    {Module_Identification.
  =     Module_Name;

Last_Transfer_Symbol : FILE
    {Contains the last Transfer_Symbol processed.
  =     Transfer_Symbol;

Linkage_Definition_Item : FLOW
  =(    Entry_Definition
    |   External_Reference);

Local_Block_ID : ELEMENT
  =     1 .. max_block_ID;

Local_Block_ID_Pair : FLOW
  =     Local_Block_ID_1
    +   Local_Block_ID_2;

Local_Block_Name : FLOW
  =     Module_Name
    +   Section_Ordinal;
```

```
CWS_Interval_Size : ELEMENT
   {The interval size for determining the working set of Block_Reference's
   {which is maintained in Block_Working_Set_List.
  =;

Defined_Entry_Point_Entry : FLOW
  =    Entry_Point_Name
   +   Entry_Location
   +   Local_Remote_Flag
   +   [Block_ID]
   +   [Instrumented_Callee_Ordinal];

Defined_Entry_Point_List : FILE
  =    <Defined_Entry_Point_Entry>;

Destination_Location : FLOW
  =    Location;

Entry_Definition : FLOW
  =    Entry_Point_Name
   +   Entry_Location;

Entry_Location : FLOW
  =    Location;

External_Reference : FLOW
  =    External_Name
   +   Reference_Type
   +   Reference_Location;

Instrumented_Callee_Entry : FLOW
  =    *Instrumented_Callee_Ordinal
   +   Entry_Location
   +   Block_ID;

Instrumented_Callee_List : FILE
  =    <Instrumented_Callee_Entry>;

Instrumented_Callee_Ordinal : ELEMENT
  =    0 .. max_callee_ordinal;

Instrumented_Procedure_Linkage : FLOW
  =    Intercept_Procedure_Linkage
   +   Instrumented_Callee_Ordinal;

Instrument_Flag : ELEMENT
  =(   'must'
   !   'need_not');

Interblock_Call : FLOW
  =    Block_ID
   +   Reference_Time;

Interblock_Reference : FLOW
   {The optional Block_ID will be present if and only if
   {Interblock_Reference_Type = 'call'.  The optional Reference_Time
   {will be present unless Interblock_Reference_Type = 'pop'.
  =    Interblock_Reference_Type
```

```
Local_Block_Name_Entry : FLOW
  =     *Local_Block_ID
  + Local_Block_Name:

Local_Block_Summary : FLOW
  =     Local_Block_Name
   +    Block_Percentage
   +    Block_Chargeback_Percentage:

Local_Remote_Flag : ELEMENT
        (Defines whether or not an entry point was declared within
        (the Target_Object_Text.
  =(    'local'
   !    'remote');

Location : FLOW
  =     Section_Original
   +    Section_Offset;

Module_Definition_Item : FLOW
  =(    Module_Identification
   !    Section_Definition);

Module_Identification : FLOW
  =     Module_Name;

Object_Text_Module : FLOW
  =     <Object_Text_Record>:

Object_Text_Record : FLOW
  =(    Module_Identification
   !    Section_Definition
   !    Entry_Definition
   !    External_Reference
   !    Address_Formulation
   !    Transfer_Symbol_Definition);

Optimal_Block_Ordering : FLOW
   (Defines the order that blocks should appear in the restructured program.
  =     <Local_Block_ID>:

Program_Data : FLOW
  =     <Byte>:

Program_Instruction : FLOW
   (Any of the instructions in the CYBER 180 CPU instruction set.
  =(    'return'
   !    'pop'
   !    'other');

Program_Profile : FLOW
   (Delineates the percentage of execution time spent in each block of
   (the program.
  =     Prog_Total_Execution_Time
  + Remote_Block_Total_Percentage
   +    <Local_Block_Summary>
   + + <Remote_Block_Summary>:
```

```
RD_Block_Ordering_Line : FLOW
  =    Local_Block_Name
   +   "..";

RD_Heading : ELEMENT
  =    "PROC Restructure  object_text_lfn, library_lfn, gen_module_name"
   +   "Create_object_library"
   +   "select_display_level  off=A"
   +   "create_module   name= gen_module_name ..."
   +   "                 first= (object_text_lfn, (..";

RD_Tail : ELEMENT
  =    "generate  library_lfn"
   +   "end";

RD_Transfer_Symbol_Line : FLOW
  =    "                 )).."
   +   "                 procedure= "
   +   Transfer_Symbol;

eference_Location : FLOW
  =    Location;

eference_Type : ELEMENT
  =(   "external_procedure"
   |   "other");

emote_Block_ID : ELEMENT
  =    1 .. max_block_ID;

emote_Block_Name : FLOW
  =    External_Name;

emote_Block_Name_Entry : FLOW
  =    *Remote_Block_ID
   +   Remote_Block_Name;

emote_Block_Percentage : ELEMENT
  (Percentage of total program execution time spent in the associated
  (remote block.
  =;

emote_Block_Summary : FLOW
  =    Remote_Block_Name
   +   Remote_Block_Percentage;

emote_Block_Total_Percentage : ELEMENT
  (Percentage of total program execution time spent in remote blocks.
  =;


emote_Total : ELEMENT
  (The total execution time spent in remote procedures due to calls
  (emanating from within a block.
  =;

structured_Program : FILE
  (A library file consisting of a single module which contains the
```

```
   {restructured blocks of the object program.

Restructure_Directives : FLOW
   {Consists of SCL commancs to invoke the library gereration utility to
   {accomplish the actual restructuring.
  =     RD_Heading
   +    <RD_Block_Ordering_Line>
   +    RD_Transfer_Symbol_Line
   +    RD_Tail;

Section_Attribute_Entry : FLOW
   {The optional Local_Block_ID will be present if and only if
   {Section_Type = 'code_section'.
  =     *Section_Ordinal
   +    Section_Type
   +    [Local_Block_ID];

Section_Attribute_Table : FILE
  =     <Section_Attribute_Entry>;

Section_Definition : FLOW
  =     Section_Ordinal
   +    Section_Type;

Section_Type : ELEMENT
  =(    'code_section'
   + 'other');

Subordinate_Time : FLOW
   {An accumulation of the execution time accrued by blocks called as
   {subordinates of a block on the Active_Block_Stack.
  =;

Target_Object_Text : FILE
  =     <Object_Text_Module>;

Target_Text_Transfer_Symbol : FILE
        {This file is used to preserve the identity of the default entry
        {point for the Target_Object_Text in its original state.  This is
        {necessary to insure that the restructured program will have the
        {same default entry point.
  =     Transfer_Symbol;

Transfer_Symbol : FLOW
  =     Program_Name;

Transfer_Symbol_Definition : FLOW
  =     Transfer_Symbol;

Transfer_Symbol_Parameter : FLOW
  =     Transfer_Symbol;

Trapped_Instruction : FLOW
  = Program_Instruction;

Unsatisfied_External_Entry : FLOW
  =     External_Name
```

```
    +   Reference_Location
    +   Instrument_Flag;

Unsatisfied_External_List : FILE
  =     <Unsatisfied_External_Entry>;

User_Address_Space : FILE
  =<    *Location
    +   (Program_Data | Program_Instruction)>;

Working_Set_Block_ID : FLOW
  =     Local_Block_ID;
```

```
Instrument_Intra_Module_Calls : PROCESS 1.1.1 =
    IF Reference_Type = 'external_procedure' THEN
        ASSIGN NEXT Instrumented_Callee_Ordinal
        GET Section_Attribute_Entry CORRESPONDING TO
                (Section_Ordinal OF Destination_Location)
        SET Entry_Location TO Destination_Location
        SET Block_ID TO (Local_Block_ID OF Section_Attribute_Entry)
        PUT Instrumented_Callee_Entry
        PUT Instrumented_Procedure_Linkage IN User_Address_Space
                AT Reference_Location
    PROCESS_END;

Save_Module_Name : PROCESS 1.1.2.1 =
    REPLACE Module_Name IN Last_Module_Name
    PROCESS_END;

Save_Local_Block_Name : PROCESS 1.1.2.2 =
    IF Section_Type = 'code_section' THEN
        GET Module_Name FROM Last_Module_Name
        ASSIGN NEXT Local_Block_ID
        PUT Local_Block_Name_Entry
    PUT Section_Attribute_Entry
    PROCESS_END;

Define_Local_Entry_Points : PROCESS 1.1.3.1 =
    SET Local_Remote_Flag TO 'local'
    GET Section_Attribute_Entry CORRESPONDING TO
        (Section_Ordinal OF Entry_Location)
    SET Block_ID TO (Local_Block_ID OF Section_Attribute_Entry)
    {NOTE...This process is dependent on the fact that, for each module,
    {       all Section_Definition's are processed prior to the first
    {       Entry_Definition.
    PUT Defined_Entry_Point_Entry
    FOR EACH Unsatisfied_External_Entry IN Unsatisfied_External_List DO
        IF Entry_Point_Name = External_Name THEN
            IF Instrumented_Callee_Ordinal ABSENT FROM
                    Defined_Entry_Point_Entry THEN
                ASSIGN NEXT Instrumented_Callee_Ordinal
                REPLACE Defined_Entry_Point_Entry
                PUT Instrumented_Callee_Entry
            PUT Instrumented_Procedure_Linkage IN User_Address_Space
                    AT Reference_Location
    PROCESS_END;

Detect_Local_Instrumentable_Calls : PROCESS 1.1.3.2 =
    IF Reference_Type = 'external_procedure' THEN
        SEARCH Defined_Entry_Point_List FOR External_Name = Entry_Point_Name
            IF MATCH THEN
                IF Instrumented_Callee_Ordinal ABSENT FROM
                        Defined_Entry_Point_Entry THEN
                    ASSIGN NEXT Instrumented_Callee_Ordinal
                    IF Local_Remote_Flag = 'remote' THEN
                        ASSIGN NEXT Remote_Block_ID
                        PUT Remote_Block_Name_Entry
                    REPLACE Defined_Entry_Point_Entry
                    PUT Instrumented_Callee_Entry
                PUT Instrumented_Procedure_Linkage IN User_Address_Space
                        AT Reference_Location
```

```
            OTHERWISE
                SET Instrument_Flag TO 'must'
                PUT Unsatisfied_External_Entry
    PROCESS_END;

Save_Local_Transfer_Symbol : PROCESS 1.1.4 =
    REPLACE Transfer_Symbol IN Last_Transfer_Symbol
    REPLACE Transfer_Symbol IN Target_Text_Transfer_Symbol
    PROCESS_END;

Save_Remote_Transfer_Symbol : PROCESS 1.2.1 =
    REPLACE Transfer_Symbol IN Last_Transfer_Symbol
    PROCESS_END;

Define_Remote_Entry_Points : PROCESS 1.2.2 =
    SET Local_Remote_Flag TO 'remote'
    PUT Defined_Entry_Point_Entry
    FOR EACH Unsatisfied_External_Entry IN Unsatisfied_External_List DO
        IF External_Name = Entry_Point_Name THEN
            IF Instrument_Flag = 'must' THEN
                IF Instrumented_Callee_Ordinal ABSENT FROM
                        Defined_Entry_Point_Entry THEN
                    ASSIGN NEXT Instrumented_Callee_Ordinal
                    ASSIGN NEXT Remote_Block_ID
                    PUT Remote_Block_Name_Entry
                    REPLACE Defined_Entry_Point_Entry
                    PUT Instrumented_Callee_Entry
                PUT Instrumented_Procedure_Linkage IN User_Address_Space
                        AT Reference_Location
    PROCESS_END;

Detect_Remote_Instrumentable_Calls : PROCESS 1.2.3 =
    IF Reference_Type = 'external_procedure' THEN
        SEARCH Defined_Entry_Point_List FOR Entry_Point_Name = External_Name
            IF MATCH THEN
                IF Local_Remote_Flag = 'local' THEN
                    IF Instrumented_Callee_Ordinal ABSENT FROM
                            Defined_Entry_Point_Entry THEN
                        ASSIGN NEXT Instrumented_Callee_Ordinal
                        REPLACE Defined_Entry_Point_Entry
                        PUT Instrumented_Callee_Entry
                    PUT Instrumented_Procedure_Linkage IN User_Address_Space
                        AT Reference_Location
            OTHERWISE
                SET Instrument_Flag TO 'need_not'
                PUT Unsatisfied_External_Entry
    PROCESS_END;

Instrument_Initial_Transfer : PROCESS 1.3 =
    IF Transfer_Symbol_Parameter ABSENT THEN
        GET Transfer_Symbol FROM Last_Transfer_Symbol
    SEARCH Defined_Entry_Point_List FOR Transfer_Symbol = Entry_Point_Name
        IF MATCH THEN
            IF Local_Remote_Flag = 'local' THEN
                IF Instrumented_Callee_Ordinal ABSENT FROM
                        Defined_Entry_Point_Entry THEN
                    ASSIGN NEXT Instrumented_Callee_Ordinal
                    REPLACE Defined_Entry_Point_Entry
```

```
                    PUT Instrumented_Callee_Entry
    PROCESS_END;

Record_Intercepted_Call : PROCESS 2.1 =
    GET Instrumented_Callee_Entry
    COPY Block_ID FROM Instrumented_Callee_Entry TO Interblock_Reference
    GET Current_Process_Time
    SET Reference_Time TO Current_Process_Time
    SET Interblock_Reference_Type TO 'call'
    PUT Interblock_Reference
    PROCESS_END;

Record_Intercepted_Return : PROCESS 2.2 =
    GET Current_Process_Time
    SET Reference_Time TO Current_Process_Time
    SET Interblock_Reference_Type TO 'return'
    PUT Interblock_Reference
    PROCESS_END;

Record_Trapped_Block_Exit : PROCESS 2.3 =
    GET Condition_Handler_Packet
    GET Trapped_Instruction FROM User_Address_Space
            AT (Program_Address OF Condition_Handler_Packet)
    CASE Trapped_Instruction CF
        ='return'=
            GET Current_Process_Time
            SET Reference_Time TO Current_Process_Time
            SET Interblock_Reference_Type TO 'return'
            PUT Interblock_Reference
        ='pop'=
            SET Interblock_Reference_Type TO 'pop'
            PUT Interblock_Reference
        ELSE
            error - hardware malfunction
    PROCESS_END;

Determine_Reference_Type : PROCESS 3.1.1 =
    FOR EACH Interblock_Reference IN Interblock_Reference_String DO
        CASE Interblock_Reference_Type OF
            ='call'=
                SEND Interblock_Call
            ='return'=
                SEND Interblock_Return
            ='pop'=
                {A count will be maintained of the number of interblock POP's
                {appearing.  When a subsequent RETURN appears, it will be
                {treated as (pop_count + 1) RETURN's, all occurring at the
                {same time.  This count must be maintained on the
                {Active_Block_Stack since a CALL/RETURN may intervene between
                {a POP and the associated RETURN.
    PROCESS_END;

Pop_Block_Statistic : PROCESS 3.1.2 =
    POP Block_Statistic FROM Active_Block_Stack
    INCREMENT Block_Total CORRESPONDING TO Block_ID BY
            Reference_Time - Call_Time - Subordinate_Time
    INCREMENT Subordinate_Time IN
            (Block_Statistic AT TOP OF Active_Block_Stack)
```

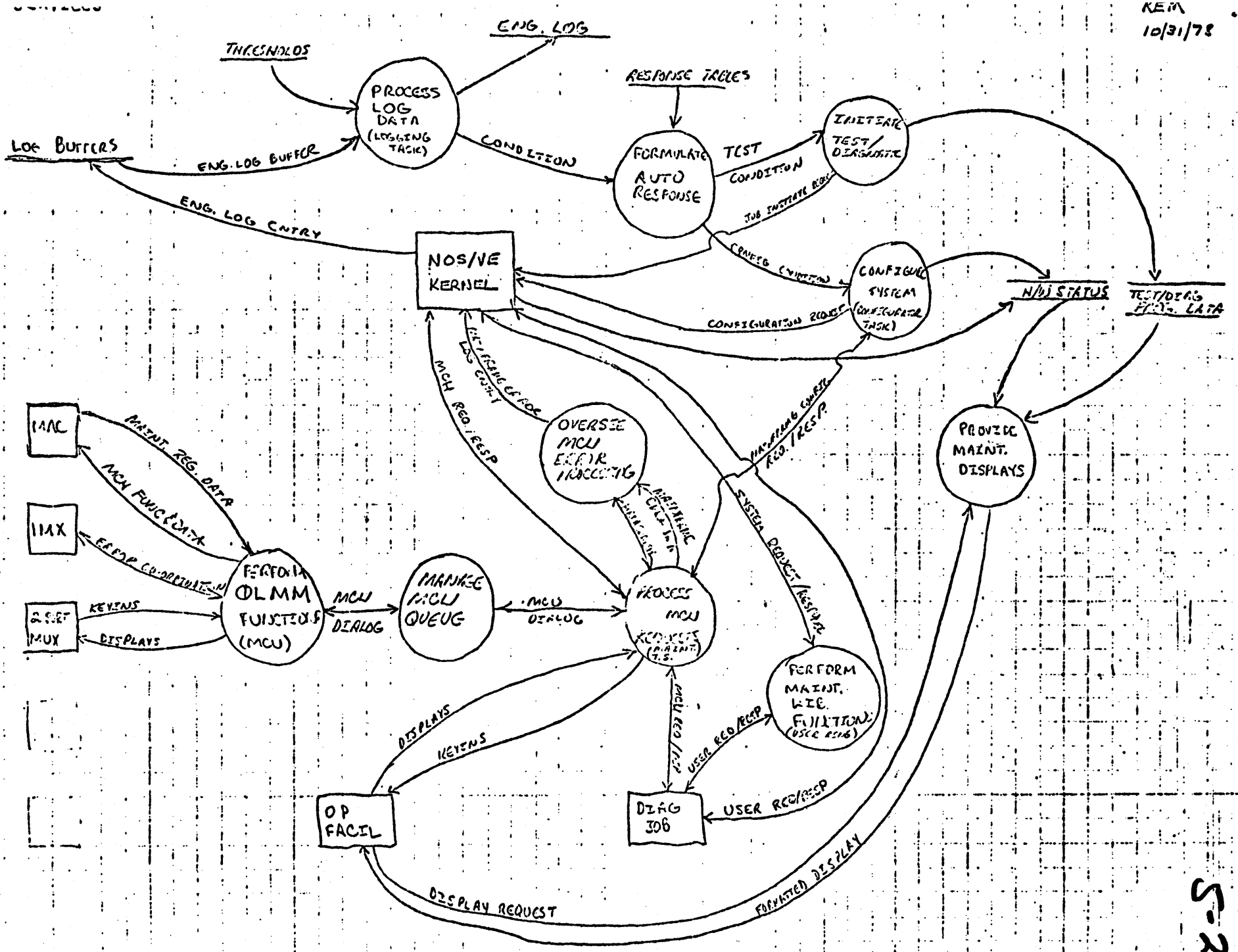
```
            BY Reference_Time - Call_Time
    IF Block_ID IS Remote_Block_ID THEN
        INCREMENT Remote_Total CORRESPONDING TO
                (Block_ID OF Block_Statistic AT TOP OF Active_Block_Stack)
                BY Reference_Time - Call_Time - Subordinate_Time
    GET Block_ID FROM (Block_Statistic AT TOP OF Active_Block_Stack)
    IF Block_ID IS Local_Block_ID THEN
        SEND Block_Reference
    PROCESS_END;

Push_Block_Statistic : PROCESS 3.1.3 =
    SET Subordinate_Time TO 0
    PUSH Block_Statistic ON Active_Block_Stack
    IF Block_ID IS Local_Block_ID THEN
        SEND Block_Reference
    PROCESS_END;

Detect_Critical_References : PROCESS 3.1.4 =
    SEARCH Block_Reference_History FOR MATCH ON Local_Block_ID
        IF NO MATCH THEN
            FLUSH Block_Working_Set
            FOR EACH Block_Reference IN Block_Reference_History DO
                {NOTE...This scan of Block_Reference_History may be
                {       implemented as part of the preceding SEARCH loop.
                SEARCH Block_Working_Set FOR
                        Working_Set_Block_ID = Local_Block_ID
                    IF NO MATCH THEN
                        SET Working_Set_Block_ID TO Local_Block_ID
                        PUT Working_Set_Block_ID
            INCREMENT Connection_Strength IN Connectivity_Matrix
                    CORRESPONDING TO (Local_Block_ID, Local_Block_ID)
                    BY 1
            FOR EACH Working_Set_Block_ID IN Block_Working_Set DO
                INCREMENT Connection_Strength IN Connectivity_Matrix
                        CORRESPONDING TO
                            (Local_Block_ID, Working_Set_Block_ID)
                        BY 1
    {NOTE...Sending Block_Reference must be the last function
    {       accomplished by this process.
    SEND Block_Reference
    PROCESS_END;

Update_Block_Reference_History : PROCESS 3.1.5 =
    WHILE (Reference_Time OF HEAD OF Block_Reference_History)
            < (Reference_Time - CWS_Interval_Size) DO
        REMOVE HEAD OF Block_Reference_History
    PUT Block_Reference AT TAIL OF Block_Reference_History
    PROCESS_END;

Assemble_Intercolumn_Bond_Matrix : PROCESS 3.2.1 =
    FOR i := 1 to number_of_local_blocks DO
        FOR j := 1 TO number_of_local_blocks DO
            SET Intercolumn_Bond_Matrix [i,j] TO 0
            FOR k := 1 TO number_of_local_blocks DO
                INCREMENT Intercolumn_Bond_Matrix [i,j] BY
                        Connectivity_Matrix [k,i] * Connectivity_Matrix [k,j]
    PROCESS_END;
```

```
Initialize_Working_Lists : PROCESS 3.2.2 =
   INITIALIZE Cluster_List TO BE EMPTY
   SET Cluster_Merit TO 0
   SET Best_Position TO 0
   FOR EACH Local_Block_ID DO
       PUT Candidate_Evaluation
   PROCESS_END;

Evaluate_Candidates : PROCESS 3.2.3 =
   FOR EACH Candidate_Evaluation IN Candidate_List DO
       SET Cluster_Merit TO
               Intercolumn_Bond_Matrix [Local_Block_ID, HEAD(Cluster_List)]
       SET Best_Position TO 0
       FOR EACH element IN Cluster_List DO
           SET test_value TO Intercolumn_Bond_Matrix [element, Local_Block_ID]
                   + Intercolumn_Bond_Matrix [Local_Block_ID, NEXT(element)]
                   - Intercolumn_Bond_Matrix [element, NEXT(element)]
           {NOTE...If element is at tail of Cluster_List then
           {       Intercolumn_Bond_Matrix [x, NEXT(element)]
           {       evaluates as 0 for all x.
           IF test_value >= Cluster_Merit THEN
               SET Cluster_Merit TO test_value
               SET Best_Position TO POSITION OF element
   PROCESS_END;

Select_Best_Candidate : PROCESS 3.2.4 =
   FIND FIRST Candidate_Evaluation IN Candidate_List
           WITH GREATEST Cluster_Merit

   {NOTE...If no Candidate_Evaluation has non-zero Cluster_Merit then the
   {       current Cluster_List represents an 'atomic' cluster, i.e., the
   {       algorithm will only append blocks to the end of the subcluster.
   {       To reduce computational complexity, the subcluster may effectively
   {       be removed from the Cluster_List at this time.

   INSERT  Local_Block_ID INTO Cluster_List AT Best_Position
   REMOVE Candidate_Evaluation FROM Candidate_List
   PROCESS_END;

Generate_Optimal_Ordering : PROCESS 3.2.5 =
   {This process consists of outputing the Local_Block_ID's contained in
   {Cluster_List in order.
   PROCESS_END;

Format_Restructure_Directives : PROCESS 3.3 =
   PUT RD_Heading
   FOR EACH Local_Block_ID IN Optimal_Block_Ordering DO
       GET Block_Name_Map_Entry
       PUT RD_Block_Ordering_Line
   GET Transfer_Symbol FROM Target_Text_Transfer_Symbol
   PUT RD_Transfer_Symbol_Line
   PUT RD_Tail
   PROCESS_END;

Build_Program_Profile : PROCESS 3.4 =
   SUM Block_Total OVER ALL Block_ID TO OBTAIN
           Prog_Total_Execution_Time
   SUM Block_Total OVER ALL Remote_Block_ID TO OBTAIN remote_total
```

```
   SET Remote_Block_Total_Percentage TO
         remote_total * 100 / Prog_Total_Execution_Time
   PUT Prog_Total_Execution_Time
   PUT Remote_Block_Total_Percentage

   FOR EACH Local_Block_ID DO
       GET Local_Block_Name_Entry
       GET Block_Execution_Totals
       SET Block_Percentage TO
                Block_Total * 100 / Prog_Total_Execution_Time
       SET Block_Chargeback_Percentage TO
                (Block_Total + Remote_Total)
                * 100 / Prog_Total_Execution_Time
       PUT Local_Block_Summary
   FOR EACH Remote_Block_ID DO
       GET Remote_Block_Name_Entry
       GET Block_Execution_Totals
       SET Remote_Block_Percentage TO
                Block_Total * 100 / Prog_Total_Execution_Time
       PUT Remote_Block_Summary
   PROCESS_END;

estructure_Program : PROCESS 4.0 =
   {This process consists of execution of the Restructure_Directives as
   {an SCL procedure.  Parameters passed to the procedure are:
   {            object_Text_lfn
   {            library_lfn
   {            gen_module_name
   {These parameters will be specified via the PMAF control statement.
   PROCESS_END;
```

5.0 MAINTAIN SYSTEM HARDWARE

KEM
10/31/78



5-2

Fold Line