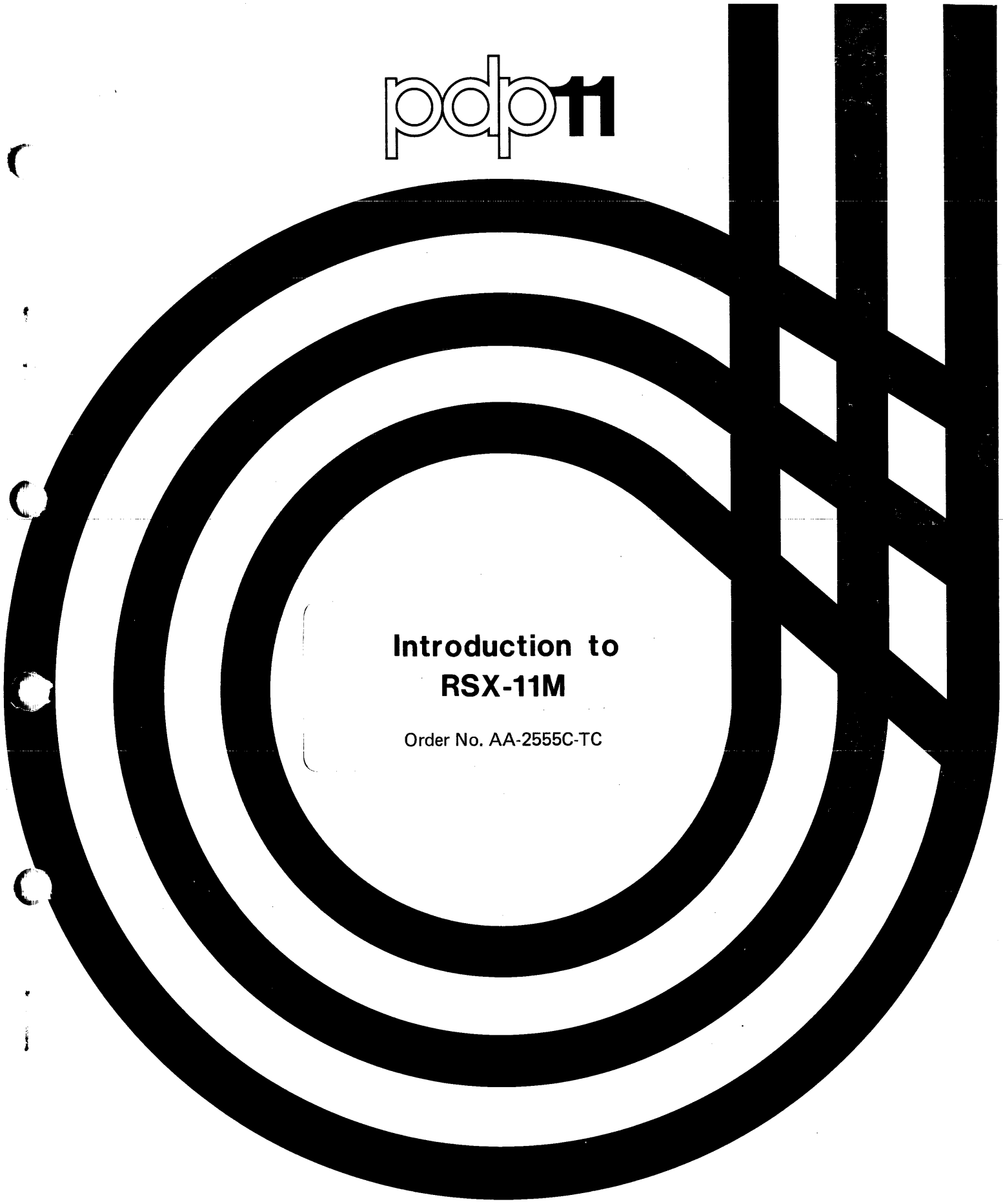

Introduction to RSX-11M

Order No. AA-2555C-TC

digital

First Printing, May 1974
Revised: November 1976
December 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

Introduction to RSX-11M

Order No. AA-2555C-TC

RSX-11M Version 3.1

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard. massachusetts

First Printing, May 1974
Revised: November 1976
December 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

			Page
PREFACE			v
CHAPTER	1	THE RSX-11M AND RSX-11S OPERATING SYSTEMS	1-1
	1.1	RSX-11M	1-1
	1.2	RSX-11S	1-1
CHAPTER	2	RSX-11M APPLICATIONS	2-1
	2.1	REAL-TIME APPLICATIONS	2-1
	2.1.1	Data Acquisition	2-1
	2.1.2	Process Control	2-2
	2.1.3	Manufacturing	2-2
	2.1.4	Laboratory and Medical Data Processing	2-2
	2.2	SUPPORTED LANGUAGES AND PROGRAM DEVELOPMENT	2-3
	2.3	COMPUTER NETWORKS	2-3
CHAPTER	3	REAL-TIME MULTIPROGRAMMING	3-1
	3.1	MEMORY ORGANIZATION	3-1
	3.1.1	Mapped and Unmapped Systems	3-1
	3.1.2	Partition Types	3-2
	3.1.3	Subpartitions	3-2
	3.2	EXECUTIVE CONTROL	3-3
	3.2.1	Task State	3-4
	3.2.2	Priority	3-5
	3.2.3	Checkpointing	3-5
	3.2.4	Distributing CPU and Memory Resources	3-6
	3.2.4.1	Round-robin Scheduling	3-6
	3.2.4.2	Swapping	3-6
	3.2.5	Significant Events	3-7
	3.2.6	Example of a 16K Unmapped System	3-7
	3.3	SYSTEM DIRECTIVE FUNCTIONS	3-8
	3.3.1	Event Flags	3-9
	3.3.2	System Traps	3-9
	3.3.3	Extended Logical Address Space	3-9
CHAPTER	4	SYSTEM OPERATION	4-1
	4.1	THE MCR INTERFACE	4-1
	4.1.1	External Scheduling of Task Execution	4-2
	4.1.2	Privileged Commands	4-2
	4.1.3	Indirect Command Files	4-2
	4.1.4	The MCR Indirect File Processor	4-3
	4.1.4.1	Symbols	4-3
	4.1.4.2	Symbol Value Substitution	4-3
	4.2	TERMINAL OPERATION	4-3
	4.2.1	Attached Terminals	4-4
	4.2.2	Slave Terminals	4-4
	4.3	MULTIUSER PROTECTION	4-4
	4.3.1	Public and Private Devices	4-4

			Page
	4.4	SYSTEM MAINTENANCE FEATURES	4-5
	4.4.1	Error Logging	4-5
	4.4.2	Diagnostic Tasks	4-5
	4.4.3	Power Failure Restart	4-6
CHAPTER	5	PROGRAM DEVELOPMENT	5-1
	5.1	PROGRAMMING LANGUAGES AND SORT UTILITY	5-1
	5.1.1	MACRO-11	5-1
	5.1.2	FORTRAN	5-1
	5.1.3	COBOL	5-2
	5.1.4	SORT-11	5-2
	5.1.5	BASIC-11	5-2
	5.1.6	BASIC-PLUS-2	5-2
	5.2	EDITING UTILITIES	5-3
	5.3	DEBUGGING AIDS	5-3
	5.3.1	Online Debugging Tool (ODT)	5-3
	5.3.2	Post-Mortem and Snap-Shot Dumps	5-4
	5.4	THE TASK BUILDER	5-4
CHAPTER	6	FILES AND I/O OPERATIONS	6-1
	6.1	FILES-11	6-1
	6.1.1	File Ownership and Directories	6-1
	6.1.2	File Protection	6-2
	6.1.3	File Specifiers	6-2
	6.1.4	File Manipulation	6-2
	6.2	TASK I/O OPERATIONS	6-3
	6.2.1	File Control Services and Record Management Services	6-3
	6.2.2	Device Independence	6-4
	6.3	PHYSICAL I/O OPERATIONS	6-4
INDEX			Index-1

FIGURES

FIGURE	3-1	Comparison of Sequential and Concurrent Execution of Programs	3-3
	3-2	Sample Unmapped System Memory Layout	3-7
	5-1	Steps in Creating a FORTRAN Task	5-5

PREFACE

0.1 MANUAL OBJECTIVES AND READER ASSUMPTIONS

The purpose of this manual is to introduce the basic concepts and capabilities of RSX-11M and RSX-11S. To understand the subjects discussed, the reader needs a general knowledge of computing terms and principles. However, the reader does not need to have a previous knowledge of either system.

0.2 STRUCTURE OF THE DOCUMENT

The manual contains 6 chapters that discuss the following topics:

Chapter 1 -- A brief description of RSX-11M and RSX-11S

Chapter 2 -- RSX-11M applications

Chapter 3 -- Memory structures and control of real-time multiprogramming

Chapter 4 -- System operation

Chapter 5 -- Program development and associated utilities

Chapter 6 -- File system and I/O operations

0.3 ASSOCIATED DOCUMENTS

The RSX-11M/RSX-11S Documentation Directory defines the intended readership of each manual in the RSX-11M/RSX-11S set and provides a brief synopsis of each manual's contents.

For detailed information about PDP-11 hardware, readers should refer to the following books:

The PDP-11 Peripherals Handbook

The PDP-11/04/05/10/35/40/45 Processor Handbook

The LSI PDP-11/03 Processor Handbook

The PDP-11/34 Processor Handbook

The PDP-11/60 Processor Handbook

The PDP-11/70 Processor Handbook

CHAPTER 1

THE RSX-11M AND RSX-11S OPERATING SYSTEMS

1.1 RSX-11M

RSX-11M is a real-time multiprogramming operating system designed for the PDP-11 series of DIGITAL computers. The design of RSX-11M is specifically geared toward quick response to real-time events; but the multiprogramming capability of RSX-11M allows real-time activity to be combined with program development and other less urgent user applications. At one extreme, RSX-11M can be used as a dedicated, stand-alone process controller; at the other, it can be a multiuser system used primarily for developing and running applications programs.

RSX-11M offers a wide range of services and utilities from which to choose. (Some of these options are automatically available as a basic part of RSX-11M; others must be purchased under separate license.) Each installation selects from these options to shape its version of RSX-11M according to the chosen hardware (PDP-11 processor and peripherals) and to the end purpose of the computer system. When an installation has decided upon the services and utilities appropriate to its requirements, the user performs a procedure called system generation to create the tailored RSX-11M system.

Every installation initially receives a minimal RSX-11M system on distribution media. This minimal, but completely operational, system is used to generate the target system in two phases. The first phase defines and assembles the Executive (see Chapter 3 for a detailed description of the Executive) by conducting a dialogue with the user. Query programs pose questions at a terminal; the answers to the questions then determine the Executive service options, processor options, and peripheral devices to be incorporated into the system. The second phase builds the Executive, allows the user to define memory structures (called partitions), and builds and installs system programs. The user then completes the process by bootstrapping and saving the new system.

When an installation decides to change its hardware or software configuration, another system generation is performed to update the version of RSX-11M.

1.2 RSX-11S

RSX-11S is a memory-only version of the disk-based RSX-11M operating system. RSX-11S can operate within a minimum of 8K words of memory (leaving 4K words for user programs), up to a maximum of 124K words. It is fully compatible internally with RSX-11M, supports an identical I/O driver interface, and is a compatible subset at the user level. Drivers written to run on one system can be incorporated and run on

the other system without change. Any nonprivileged user program that executes under RSX-11S V2.1 can execute under RSX-11M V3.1 without change, following a re-link of the object program. The same holds true for RSX-11S V2.1 and RSX-11D V6.2, so long as the program does not use memory management directives.

Because RSX-11S is a memory-only system, it does not support a file system, nonresident tasks, checkpointing, or program development. Furthermore, it does not support dynamic memory allocation of system-controlled partitions in a mapped system. Its main purpose is to provide a runtime environment for the execution of tasks on a processor that has a modest complement of peripheral devices.

The user interface to the RSX-11S system is a subset of the RSX-11M Monitor Console Routine (MCR) called Basic MCR. Strict subset compatibility is provided.

Two additional privileged tasks are available for inclusion in an RSX-11S system. The Online Task Loader (OTL) is used to install, load, and fix tasks in a system. The System Image Preservation program (SIP) saves an image of an operational system for subsequent bootstrapping.

Program development and system generation for an RSX-11S system require a host RSX-11M V3.1 or RSX-11D V6.2 system. Tasks may be written, assembled or compiled, linked on the host system, and then transported to an RSX-11S system for execution.

CHAPTER 2

RSX-11M APPLICATIONS

This chapter discusses both general and specific RSX-11M applications. It illustrates several real-time jobs that can be handled by RSX-11M, and it introduces two important features that affect RSX-11M's potential uses. These two features are RSX-11M's program development capability and network communications.

Real-time events often demand immediate and full access to available resources at regular and irregular intervals. Between peak load times, the computer resources can be used to run less urgent jobs, such as program development, payroll processing, and offline data reduction. By means of the mechanisms described in Chapter 4, RSX-11M can pre-empt the less urgent programs whenever it needs to service a real-time event.

RSX-11M can manage the concurrent operation of both real-time and less time-dependent programs. This capability means that an RSX-11M installation can use idle time to its best advantage. In fact, non-real-time applications can comprise the major work load of an RSX-11M system.

2.1 REAL-TIME APPLICATIONS

Real-time applications are specifically those jobs that require response to physical events as they occur.

2.1.1 Data Acquisition

Data acquisition is the collection of physically generated data for subsequent use in evaluation and control. A typical data acquisition application requires several user programs to respond concurrently to bursts of data. RSX-11M multiprogramming allows the concurrent execution of user programs required to service such data acquisition applications. For example, RSX-11M, under user program direction, can process data acquired from instruments such as spectrometers, flowmeters, and thermocouples.

RSX-11M provides a rapid response time to external stimuli because the system or user-written programs that control peripheral devices are linked directly to the PDP-11 hardware interrupt vectors.

2.1.2 Process Control

To control the behavior of a continuous process, a process control application usually involves data acquisition, analysis, and a feedback loop. Oil refineries and steel rolling mills are examples of industries that require such processes.

The signals that the PDP-11 receives from devices attached to the process are called process inputs. By reading and operating on various process inputs, RSX-11M software can monitor the status of many parts of the process, such as temperatures, flow rates, and the amount of raw materials being used. Engineering and operational data stored in the system can be used to determine what action should be taken to keep the process running properly. Further decisions can be made by control optimization programs, which make adjustments based on interrelationships of various parts of the process. After operational decisions have been made, RSX-11M, directed by user programs, generates signals that control the valves, switches, and relays that in turn control the process. These signals are called process outputs.

2.1.3 Manufacturing

In a manufacturing environment, application tasks running under RSX-11M can monitor manufacturing operations, test the quality of products, furnish production data to higher level production and inventory control systems, and manage the flow of work between departments. Data in these applications can be collected directly from sensors or entered by operating personnel into special devices located in the manufacturing plant.

2.1.4 Laboratory and Medical Data Processing

Laboratory settings usually require a small computer system capable of performing one or more precisely defined real-time activities. Possible laboratory applications include:

- Measurement of X-ray diffraction
- Gas chromatography
- Psychology experiments such as stimulus presentation
- Particle acceleration

Many medical applications have the same requirements as other laboratory settings. Patient monitoring, for example, demands instant response to any changes in the patient's condition. A PDP-11 connected to monitoring devices can keep a continual log of the patient's condition; and user-written programs can analyze the recorded data as an aid to diagnosing the symptoms.

RSX-11M can run in the small machines (16K or 24K minimum, leaving 8K for user programs) suitable to a laboratory environment. It also provides the necessary rapid response to real-time events and the program development facilities needed to create specialized software systems.

2.2 SUPPORTED LANGUAGES AND PROGRAM DEVELOPMENT

The real-time activities described above require specialized software systems. RSX-11M supports several programming languages so that the user can choose those languages most suited to particular applications. These languages are:

- MACRO-11 Assembler
- FORTRAN IV or FORTRAN IV-PLUS
- COBOL
- BASIC-11
- BASIC-PLUS-2

RSX-11M is particularly suited to environments that are subject to changing requirements. Its range of supported languages allows an installation both to improve existing applications and to develop new types of applications without changing operating systems. (Chapter 5 contains a brief description of the languages.) For example, an installation that uses RSX-11M to perform a real-time process control application might decide to use its system to handle payroll as well. All the real-time programs have probably been written in FORTRAN IV, which is not ideal for commercial applications. With little impact on the existing system, the installation might purchase RSX-11M's COBOL compiler and start developing payroll applications. The installation has been able to increase substantially the function of its RSX-11M system simply by adding on one of RSX-11M's many optional software features.

Another reason for RSX-11M's adaptability to changing requirements is its multiprogramming capability. RSX-11M users enter source programs and data from terminals. Because RSX-11M supports the concurrent operation of terminals, as well as multiprogramming, an installation can continually update and enhance its applications by running multiple-user program development hand in hand with the applications themselves.

2.3 COMPUTER NETWORKS

A computer network is an interconnection of computer systems, communication devices, and I/O devices. The computer systems, called "nodes," are connected together by intercommunication hardware, called "physical links." The nodes can be geographically remote from each other; each is capable of performing applications-oriented functions.

An RSX-11M installation can purchase a DIGITAL software package called DECNET-11M to connect its computer system to others to form a network. (DECNET is also available with RSX-11S.) DECNET-11 is the network software provided for the RSX-11 operating systems.

DECNET-11 provides a basic intertask communication capability. A task executing in one node can create a data path to a task executing in the same or another node. While the tasks exchange data over the data path, the DECNET-11 software oversees the exchange, performing functions such as error detection. In addition, DECNET-11 provides facilities that allow:

- Device sharing. Device sharing allows a DECNET-11 user to connect to and use the peripheral devices of a remotely located system.

- File sharing. File sharing permits the reading from, writing to, and updating of files on a remotely located system.

- Program sharing. A loadable program can be transferred to a remotely located system to be loaded and executed by that system.

To provide these functions, DECNET-11 is designed as a layered structure of protocols, which are formal sets of conventions governing the format, control, and sequencing of message exchange.

CHAPTER 3

REAL-TIME MULTIPROGRAMMING

Real-time multiprogramming requires the precisely organized interaction of the following major elements:

- The Executive. The RSX-11M Executive is the kernel of the operating system, the software that directs program execution.
- Programs. A program is a sequence of instructions that directs the computer in manipulating data to achieve some goal. In RSX-11M, executable programs are called "tasks." Tasks can either be user-written or supplied by DIGITAL.
- Memory. Memory is the hardware storage medium in which programs actually reside and execute.

This chapter introduces the mechanisms related to memory and the Executive that together produce the RSX-11M real-time multiprogramming environment. Section 3.1 describes how memory is organized. Section 3.2 discusses the role of the Executive. Program-related aspects are considered in parts of both Sections 3.1 and 3.2, as well as in Chapter 4. Section 3.3 discusses various system directives that are part of the RSX-11M multiprogramming scheme.

3.1 MEMORY ORGANIZATION

A task runs in a predetermined contiguous area of memory called a partition. A partition has the following characteristics:

- A name
- A defined size
- A fixed base address
- A defined type

The relationship between a task and the partition in which it runs depends on whether the system is mapped or unmapped.

3.1.1 Mapped and Unmapped Systems

RSX-11M is designed to run on almost all models of the PDP-11 processor. The PDP-11 addressing mechanism allows a program to address directly only 32K words of memory. To use memories larger than 32K words, DIGITAL has a special hardware unit available with the PDP-11/34/35/40/45/50/55/60/70 processors. This special hardware,

called the KT11 memory management unit, associates addresses expressed in programs ("virtual" addresses in the range 0 to 32K) with actual locations in memory (called "physical" addresses). Physical addresses can range from 0 to 124K words on all processors other than the PDP-11/70. Physical addresses on a PDP-11/70 can range from 0 to 1920K words.

A PDP-11 system that includes a KT11 memory management unit is called a "mapped" system. Systems without a KT11 are "unmapped."

Whether a system is mapped or unmapped affects the way in which RSX-11M users create tasks. Before a compiled program can run, it must be processed by the Task Builder, a DIGITAL-supplied program that produces the program unit, called a task, that actually runs in memory (see Section 5.4 for further details). If a system is unmapped, users must specify to the Task Builder the base address of the partition in which a task is to run. The resulting task cannot run in a partition that has a base address different from the address specified to the Task Builder.

In a mapped system, however, every task (other than a privileged task mapped into the Executive) has a virtual base address of 0. Transparently to the user, the KT11 maps the virtual addresses of a task to the actual physical addresses in which the task resides. A task in a mapped system can therefore run in any partition large enough to contain it.

3.1.2 Partition Types

RSX-11M supports two types of partitions in which tasks can execute:

1. System-controlled.

2. User-controlled.

In a system-controlled partition, the Executive allocates available space to accommodate as many tasks as possible at any one time. This allocation may involve shuffling resident tasks to arrange available space into a contiguous block large enough to contain a requested task. Only mapped systems support system-controlled partitions.

A user-controlled partition, however, is exclusively allocated to one task at a time. This type of partition is supported by both mapped and unmapped systems.

3.1.3 Subpartitions

A user-controlled partition may be subdivided into as many as seven nonoverlapping subpartitions. Like its parent main partition, a subpartition can contain only one task at a time. Since the subpartitions occupy the same physical memory as the main partition, tasks cannot be simultaneously resident in both the main partition and one or more subpartitions. But since all the subpartitions can each contain a task, up to seven tasks can potentially run in parallel within a pre-empted main partition.

The purpose of subpartitioning is to reclaim large storage areas in unmapped systems. For example, when a large task that requires a main partition is either no longer active or can be pre-empted (checkpointed), subpartitioning allows the partition space to be used by a number of smaller real-time tasks.

3.2 EXECUTIVE CONTROL

The Central Processing Unit (CPU) is the part of the computer that actually executes instructions; only one task at a time can have control of the CPU. Multiprogramming is possible because task operation almost always involves more than CPU usage. A real-time task that initiates a process and then waits for the process to complete might not use CPU time while it is waiting. Therefore, while one task waits for an event (for example, an I/O operation or a real-time process) to complete, the Executive gives control of the CPU to another task.

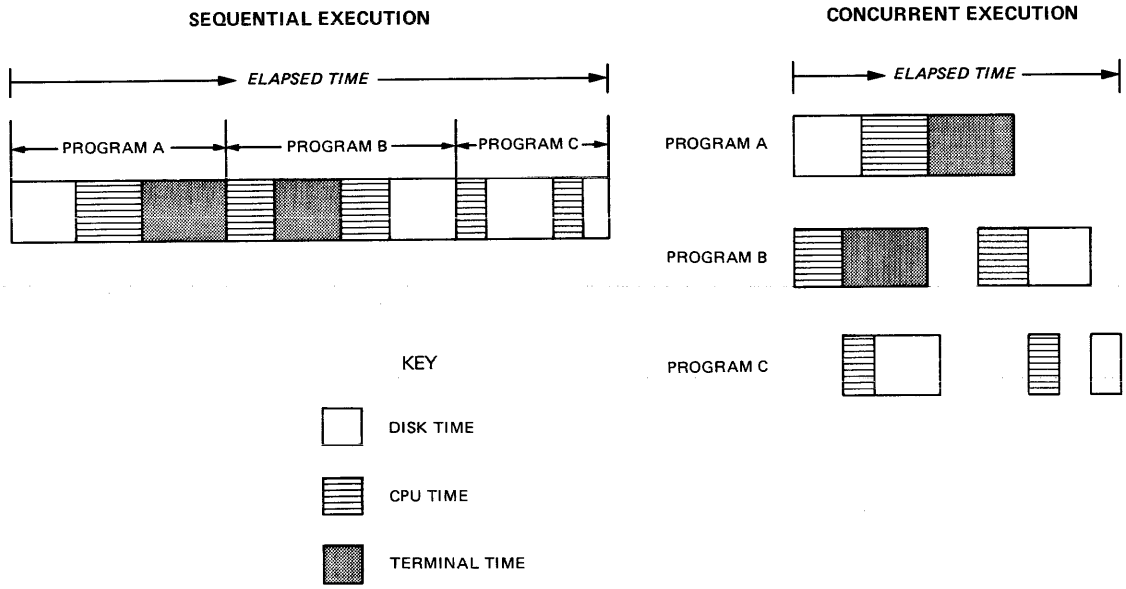

Figure 3-1 Comparison of Sequential and Concurrent Execution of Programs

Figure 3-1 illustrates the advantages of multiprogramming. When tasks A, B, and C run in a system that does not support multiprogramming, they must run one after the other. Task A reads some information from disk, operates on it, and displays a report. Task B performs some computation, displays a message, performs some more computation, and writes the result to disk. Task C performs some computation, reads some information from disk, performs some more computation, and writes the result to disk. While one part of the system, such as the disk drive, is busy, the other parts, such as the CPU, are idle.

A quite different sequence of operations is possible, however, if the three tasks run concurrently in a multiprogramming system such as RSX-11M. The three resources shown (CPU, disk drive, and terminal) are often simultaneously active; and the concurrent execution of the three tasks takes less overall elapsed time than the sequential execution of the same tasks.

In RSX-11M, the Executive coordinates the execution of all tasks resident in memory to achieve both efficient use of system resources and rapid response to real-time demands. The following criteria affect the way this coordination works:

- Task state
- Priority
- Checkpointing
- Round-robin scheduling
- Swapping
- Significant Events

3.2.1 Task State

A task is made known to the Executive by installing it. When a user installs a task (by issuing a command from a terminal), the system records a number of task parameters in a system-resident table called the System Task Directory (STD). The recorded parameters include the name and size of the task, the disk address at which the task's image starts, and the name of the partition in which the task is to run.

An installed task is defined as a task that has an entry in the STD. It is neither resident in memory nor competing for system resources. The Executive considers it to be dormant until a running task or a command issued from a terminal requests the Executive to activate it. The Executive therefore recognizes two task states:

- Dormant. A dormant task is one that has been installed (has an entry in the STD), but has not been requested to run.
- Active. An active task is an installed task that has been requested to run. It remains active until it exits, terminates, or is aborted, when it returns to dormant state.

  An active task can be in one of two substates, ready-to-run or blocked.

  1. Ready-to-run. A ready-to-run task competes with other tasks for CPU time on the basis of priority. The highest priority ready-to-run task obtains CPU time and thus becomes the current task.

  2. Blocked. A blocked task is unable to compete for CPU time for synchronization reasons or because a needed resource is not available.

The distinction between dormant tasks and active tasks is important in a real-time system. A dormant task uses little memory; and yet when the task is needed to service a real-time event, the Executive can quickly and efficiently introduce it into active competition for system resources. An installed task's STD entry enables this quick response because it contains all the parameters the system needs to retrieve the requested task. Note that the number of installed, dormant tasks can, and usually will, far exceed the number of active tasks.

When the Executive receives a request to activate a dormant nonresident task, it performs a series of actions:

- It allocates required memory resources.
- It brings the task into memory (if there is space available in its partition).
- It places the task in active competition for system resources with other resident tasks.

If the partition in which a task is installed is fully occupied and no resident task can be checkpointed (see Section 3.2.3), the task is placed in a queue by priority with other activated tasks, each waiting for space to become available in its partition.

3.2.2 Priority

Active tasks compete for system resources on the basis of priority and resource availability. The priority of a task is determined by a number assigned to the task when it is built, when it is installed, or when it is running. The number is in the range 1 to 250 (decimal), where a higher number indicates a higher priority. The highest priority task that has access to all the resources it needs has control of the CPU. When that task becomes blocked (to wait for an I/O transfer to complete, for example), the Executive looks for another task to use the CPU. The chosen task is the one that has the highest priority and has access to all the resources it needs.

In an RSX-11M installation that mixes real-time applications with less urgent work, the real-time tasks should have higher priority numbers assigned. This arrangement would ensure that the Executive gives CPU time to the real-time tasks ahead of the others.

Text editors, which are commonly used in program development systems, spend a large percentage of their runtime waiting for terminal I/O to complete and are therefore out of competition for CPU time. However, when the I/O completes, the terminal user wants a rapid response. To gain the rapid response desired, the system manager can assign to the text editors a priority that is higher than more CPU-bound tasks like the Task Builder or the Assembler.

3.2.3 Checkpointing

Checkpointing provides a means by which tasks not currently resident in memory can gain access to the CPU. In some instances, an activated task cannot compete for the processor because the partition in which it was installed is fully occupied. If the partition contains a task that has a lower priority and is checkpointable, the Executive can move that task out of memory to disk to make room for the higher priority task. When the latter task is finished, the pre-empted task is reloaded and continues processing from the point at which it was interrupted. This roll-out, roll-in process is called checkpointing.

When a task can be checkpointed, checkpoint space equal to the size of the task (or the size of the partition that contains the task) must be available on disk. (Checkpoint space contains the rolled-out task while a higher priority task executes.) Checkpoint space is allocated either statically at task build, or dynamically at runtime. The RSX-11M user can use both kinds of checkpointing to balance the advantages and disadvantages of the different allocation methods.

When submitting a compiled program to the Task Builder, the user can request that checkpoint space be allocated in the task image file. (The task image file contains the executable task.) While the task is running, its checkpoint space is always available on disk, whether or not the Executive actually checkpoints the task.

Dynamic allocation of checkpoint space allows a more efficient use of disk storage. Instead of reserving disk space equal to the size of each checkpointable task, the user can create one or more checkpoint files on disk to contain all checkpointed tasks. The size of the files depends on an estimation of the checkpoint space required at any given time. When the system allocates checkpoint space dynamically, tasks do not need to be built as checkpointable. Instead, the user decides if a task can be checkpointed when the task is installed. The user creates a checkpoint file, independently of individual tasks, by issuing a command from a terminal. Then, when the Executive needs to checkpoint a task, it rolls the task out to available space in a checkpoint file. A drawback to dynamic allocation of checkpoint space is that space in a checkpoint file may not always be available.

3.2.4 Distributing CPU and Memory Resources

The Executive allocates CPU and memory resources on the basis of priority. Two system generation options, round-robin scheduling and swapping, influence the allocation of resources when numerous active tasks have equal or similar priorities.

3.2.4.1 **Round-robin Scheduling** - When numerous competing memory-resident tasks have equal priorities, the Executive tends to give CPU time more often to those tasks that appear first in the STD queue. (Entries with equal priorities in the STD normally appear in the order in which the tasks were installed.) To avoid this problem, RSX-11M provides a system generation option called round-robin scheduling. When round-robin scheduling has been incorporated, a scheduling algorithm periodically rotates tasks of equal priority within the STD. The overall effect is to distribute use of the CPU more evenly, because each equal priority task has its turn toward the head of the queue.

3.2.4.2 **Swapping** - Another problem arises when several active tasks with similar or equal priorities are competing for partition space in memory. A task cannot normally cause the Executive to checkpoint a task with the same or a higher priority. Circumstances can therefore prevent the task of equal or lower priority from accessing memory.

Swapping is a variation of checkpointing that enables the Executive to checkpoint tasks with similar priorities in and out of memory. (The tasks must be checkpointable.) When an eligible task begins to run, the Executive adds a "swapping priority" to the task's normal running priority. The Executive then decrements the swapping priority (which eventually has a negative value) as the task runs. If the sum of the decremented swapping priority and the task's running priority causes the running task to have an actual priority less than that of a competing task, the Executive checkpoints the running task to make room for the competing task. The Executive then places the rolled-out task at the end of the queue of active tasks competing for memory. (The swapping priority does not affect CPU scheduling or I/O dispatching, which are governed solely by the task's running priority.)

3.2.5 Significant Events

Certain occurrences called significant events cause the Executive to re-evaluate the eligibility of all active tasks to run. When a significant event occurs, the Executive scans the queue of active tasks and runs the highest priority ready-to-run task. Significant events include the following:

- An I/O completion
- A task exit
- The removal of an entry from a clock queue
- The execution of one of several system directives issued by a task
- The execution of the round-robin scheduling algorithm

3.2.6 Example of a 16K Unmapped System

Figure 3-2 illustrates the memory layout of a sample 16K unmapped system. The Executive region, requiring 8K, consists of the Executive and the user-controlled main partition named SYSPAR. This partition contains the file system (F11ACP), the Monitor Console Routine (MCR), and the Task Termination Notification routine (TKTN). The file system is checkpointable and has a lower priority than MCR or TKTN. Therefore, if the file system is running and an operator requests MCR, the File System will be checkpointed, and MCR will be loaded and initiated.

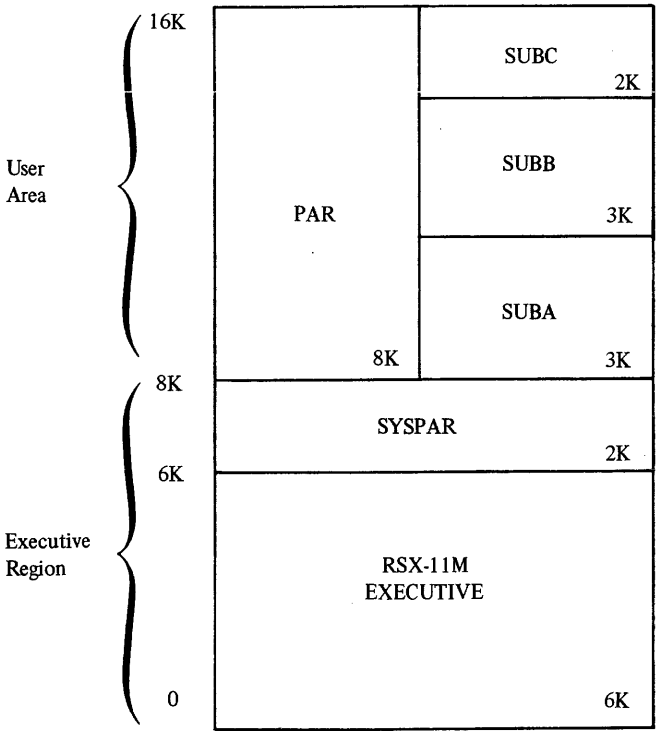

Figure 3-2 Sample Unmapped System Memory Layout

The user area is composed of a user-controlled main partition named PAR8K, 8K in length, and three subpartitions, named SUBA, SUBB, and SUBC. The 8K partition is used for program preparation tasks such as the language processors and the Task Builder. These programs usually have a low priority and may be checkpointable.

The three subpartitions are available for real-time tasks. Any time one of the real-time tasks needs the processor and has a higher priority than the task occupying the main partition, the contents of the main partition will be checkpointed (if the occupant task is checkpointable).

If the partitions SUBA, SUBB, SUBC, and SYSPAR are occupied and the tasks in them are ready to run, to which task does the Executive give the CPU? The choice depends entirely on task priority, even for the SYSPAR partition. Hence, the highest priority task will run next, regardless of where it resides in memory.

3.3 SYSTEM DIRECTIVE FUNCTIONS

A system directive is a request from a task to the Executive to perform an indicated operation. A programmer uses system directives to control the execution and interaction of tasks; and the execution of certain directives causes significant events to occur. In fact, most significant events are caused, either directly or indirectly, by the issuance of system directives. In this way, system directives are a part of RSX-11M's multiprogramming scheme. Directly or indirectly, they affect the way the Executive shares system resources among simultaneously active tasks.

System directives enable tasks to perform functions such as the following:

- Obtain task and system information
- Measure time intervals
- Perform I/O functions
- Manipulate a task's logical and virtual address space
- Suspend and resume execution
- Request the execution of another task
- Exit

System directives allow tasks to exploit some major system functions, including the following:

- Event flags
- System traps
- Extended logical address space

These three system functions are discussed in Sections 3.3.1, 3.3.2, and 3.3.3.

3.3.1 Event Flags

Significant events not only affect Executive management of task execution; the tasks themselves can also use significant events to coordinate internal task activity and to communicate with other tasks. For example, a task can issue a system directive to associate an event flag with a specific significant event. When that event occurs, the Executive sets the associated flag. Therefore, by testing the state of the flag, a task can determine whether or not the event has occurred.

Sixty-four event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding event flag number. The first 32 flags are local to each task and are set or cleared as a result of each task's operation. The second 32 flags are common to all tasks and are therefore called common flags. Common flags can be set or cleared as a result of any task's operation. Tasks use common flags to communicate with other tasks because one task cannot refer to another task's local flags.

3.3.2 System Traps

System traps are transfers of control (also called software interrupts) that provide tasks with another means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two distinct kinds of system traps:

- Synchronous System Traps (SSTs). SSTs detect events directly associated with the execution of program instructions. They are "synchronous" because they always recur at the same point in the program when previous instructions are repeated. For example, an illegal instruction causes an SST to occur.

- Asynchronous System Traps (ASTs). ASTs detect significant events that occur "asynchronously" to the task's execution; that is, the task has no direct control over the precise time that the event occurs. For example, the completion of an I/O transfer causes an AST to occur.

A task that uses the system trap facility issues system directives that establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap occurs, the task automatically enters the appropriate routine (if its entry point has been specified).

3.3.3 Extended Logical Address Space

An RSX-11M task specifies an address in a 16-bit word. The largest address that can be expressed in a 16-bit word is 65,536 bytes or 32,768 words (commonly referred to as 32K words). A task can therefore only directly address up to 32K words. To avoid limiting the size of a task to its addressing capability, a task can use overlays that are defined at task build, or it can use memory management directives.

An overlaid task is arranged into segments: a single root segment, which is always in memory, and overlay segments, which can be read into memory as required. The segments concurrently in memory cannot exceed 32K words.

Memory management directives allow task segments resident in memory to exceed 32K words. The directives use the KT11 hardware (see Section 3.1.1) to map task addresses to different logical areas within the task. Instead of displacing task segments in memory, the task can reside entirely in memory and map its virtual addresses to different physical addresses.

RSX-11M defines three kinds of address space:

- Physical address space. Physical address space consists of the physical memory in which tasks reside and execute.
- Logical address space. Logical address space is the total amount of physical address space to which the task has access rights.
- Virtual address space. Virtual address space corresponds to the 32K of addresses that the task can explicitly specify in a 16-bit word. If a task does not use memory management directives, its logical and virtual address space directly correspond. Using these directives, however, the task can map its virtual addresses to different parts of its logical address space. The net effect is to allow a task's logical address space to exceed 32K.

The memory management directives also allow a task to expand its logical address space dynamically. In other words, a task can access logical areas that are not part of its static task image (the executable task produced by the Task Builder). A task can issue directives that create a new region of logical space and then map a range of virtual addresses to the newly created region. A task can also map its virtual addresses to logical areas that belong to another task. The mapped area then becomes part of the former task's logical address space.

One result of the ability to map virtual addresses is an increased potential for task interaction by means of shared regions. For example, at runtime a task can create a new region of logical space, into which it writes a large amount of data. Any number of tasks can then access that data by mapping a range of their virtual addresses to the region. Another result is an ability to use a greater number of common routines. Tasks can simply map to required routines at runtime, rather than link to them at task build.

CHAPTER 4

SYSTEM OPERATION

RSX-11M operating procedures vary from one installation to the next, depending on the function of the computer system and its hardware configuration. A laboratory may have a PDP-11/10 dedicated to one real-time application, with a single disk and only one terminal, which is sometimes used for limited program development. A lab technician might oversee the computer system as one of many laboratory duties. In contrast, a manufacturer may have a PDP-11/70 that performs a process control job, all the company's accounting and payroll applications, as well as substantial program development. Such an installation could have a dozen or more terminals and numerous disk and magnetic tape drives. An installation this size requires a full-time operator to maintain the equipment and to manage the company's data processing mix.

Although an RSX-11M installation may not employ a person to attend to the needs of the computer system, someone must know how to operate RSX-11M. This chapter discusses basic RSX-11M operating concepts. The term "operator" refers to anyone who interfaces with or oversees RSX-11M. Most operating concepts apply to all installations; however, some features (multiuser protection, for example) are optional.

4.1 THE MCR INTERFACE

An operator communicates with RSX-11M by entering special commands at a terminal. The terminal directs the special commands to the Monitor Console Routine (MCR) processor. The MCR processor either executes the commands itself, or it activates a system or user-written task that can service the commands.

MCR commands allow an operator to:

- Start up the system
- Manage peripheral devices
- Control task execution
- Obtain system and task information
- Activate system or user-written tasks that request input from the terminal

The MCR commands that control task execution are particularly significant to system performance. An MCR command must be used to install a task into the system (see Section 3.2.1). Therefore, an operator establishes the base of installed tasks, which the Executive,

active tasks, and further MCR commands manipulate to fulfill the purpose of the RSX-11M system.

4.1.1 External Scheduling of Task Execution

An important MCR function is the external scheduling of task execution. This type of scheduling works in conjunction with the Executive's priority driven internal scheduling of active tasks. The operator can include time parameters with the command that activates an installed task. The time parameters request the Executive to run a task:

- At a specified time from the current moment
- At a specified time from clock unit synchronization
- At an absolute time of day
- Immediately

All of these time options are available with or without periodic rescheduling. RSX-11M also supports an unlimited number of programmed timers for each task in the system. The user task can create its own timer, which the Executive then decrements at regular intervals. When the timer reaches zero, the Executive sets an event flag or generates an Asynchronous System Trap (AST) that passes control to the task at a prespecified address.

4.1.2 Privileged Commands

To restrict the use of commands that directly affect system performance, RSX-11M considers some MCR commands and command options to be privileged. An operator can issue a privileged command only from a privileged terminal. Section 4.2 describes how a terminal attains privileged status.

4.1.3 Indirect Command Files

An indirect command file is a text file containing a list of commands exclusive to, and interpretable by, a single task. The interpreting task is usually a system-supplied component of RSX-11M, such as MCR, the MACRO-11 Assembler, or the Task Builder.

To initiate indirect files, the user replaces a command string with a file specifier preceded by an at sign (@). The task requesting input then accesses the specified file and reads and responds to the commands contained within it. For example, to initiate a file of MACRO-11 commands the user enters the following:

```
MAC  @INPT.CMD
```

The MACRO-11 Assembler then accesses the file INPT.CMD for its commands.

Indirect files can reference other indirect files, but the maximum nesting depth is limited to four levels of files.

4.1.4 The MCR Indirect File Processor

Most tasks read and respond to commands contained in an indirect file as if the commands were entered directly from a terminal. MCR, however, has an indirect file processor that interprets indirect commands. An MCR indirect command file can contain both MCR commands and special commands (or directives) to be interpreted by the indirect file processor itself. MCR optionally displays on the requesting terminal every command retrieved from an MCR indirect command file.

The indirect file processor first reads the command file and interprets each command line either as a command to be passed to MCR or as a request for action by the processor itself. Processor directives are distinguished by a period (.) as the first character in the line. MCR commands have no special prefix characters.

4.1.4.1 **Symbols** - The MCR indirect file processor enables the user to define symbols, which can then be tested or compared to control flow through the indirect command file. The symbols themselves consist of 1 to 6 ASCII characters. The assigned value that defines a symbol can be a true or false value, an octal or decimal number, or a character string. The first directive to specify a symbol defines the symbol; subsequent references test, compare, or redefine it.

4.1.4.2 **Symbol Value Substitution** - The symbols described above are used in directives to the indirect file processor. Indirect MCR commands can use the values assigned to string symbols by replacing a normal parameter (for example, a device-unit) with the symbol name enclosed in single quotation marks (for example, 'DEV'). When a previous directive has enabled "substitution mode," the indirect file processor replaces the symbol name enclosed in single quotation marks with the string value assigned to the symbol. When the processor encounters a single quote, it treats the subsequent text, up to the second single quote, as a symbol. The processor then searches the table of symbols for the corresponding string value, and substitutes it in the MCR command line before the line is interpreted.

4.2 TERMINAL OPERATION

In RSX-11M, a variable number of terminals can operate concurrently. In addition, each terminal operates independently of others in the system so that each can run a different task. In a system that supports multiuser protection (see Section 4.3), a user must log onto a terminal before issuing further commands. In other RSX-11M systems, a user can issue commands whenever the terminal displays an appropriate prompt.

Section 4.1.2 explains that a user can issue privileged commands only at a privileged terminal. In multiuser protection systems, individual users are either privileged or nonprivileged; when a user logs on, the terminal assumes the privilege status of that user. In other RSX-11M systems, however, a terminal's privilege status is determined at system generation. Subsequent to system generation, a user can issue an MCR command at a privileged terminal to modify the privilege status of any other terminal connected to the system.

4.2.1 Attached Terminals

RSX-11M allows tasks to request or "solicit" input from a terminal. To ensure that a soliciting task receives input intended for it, the task usually attaches to the terminal. While the task is attached, the terminal directs all input to the attached task, with one exception. The exception is a control C character (the C key typed while pressing the CTRL key), which gains the attention of the MCR processor. An attached terminal ensures that a soliciting task properly receives its input; but it also allows a user to interrupt the task's control of the terminal to communicate with MCR. Note that attaching to the terminal is a function of the task rather than of a user.

Some applications may require that a user be denied access to MCR. In this case, a task can attach to the terminal with a special subfunction. The subfunction causes the system to generate an AST for the attached task whenever someone enters unsolicited input, including CTRL/C, at the terminal.

4.2.2 Slave Terminals

When an installation needs to dedicate a terminal exclusively to one or more tasks, an operator issues an MCR command (or a task issues a special I/O function) that sets the terminal to slave status. The difference between a slave terminal and an attached terminal is that the system ignores all unsolicited input, including CTRL/C, that is entered at a slave terminal. Until the operator issues another MCR command to delete the slave status, the terminal can only be used to communicate with the task soliciting input from the terminal. An I/O function issued by a task can also delete the slave status of a terminal. Slave terminals are often dedicated to real-time applications.

4.3 MULTIUSER PROTECTION

Multiuser protection, a system generation option, allows an RSX-11M installation to monitor and control individual users of the system. As Section 4.2 states, individual users are either privileged or nonprivileged. A system manager assigns a User Identification Code (UIC) to each user, which determines the user's privilege status. When logging onto a terminal, the user supplies a last name or UIC and a password. If the user gives a name, the system finds the associated UIC. The system then checks that the password matches the last name or UIC, and sets the terminal to privileged or nonprivileged status, according to the user's UIC.

4.3.1 Public and Private Devices

Multiuser protection systems allow nonprivileged users to issue some MCR commands that are considered privileged in systems that do not include the option. There are also several "multiuser" MCR commands provided that do not exist in other versions of RSX-11M. These commands allow any user to allocate a device (a disk drive, for example) as the user's private device; allocating the device prevents other nonprivileged users from accessing it.

A nonprivileged user can access a private device to perform MCR functions that are privileged in non-multiuser systems. These functions include preparing a disk or magnetic tape for use by the RSX-11M file system and putting the disk or tape online and offline.

To complement the private device feature, multiuser protection allows a privileged user to declare certain devices to be "public." Public devices cannot be allocated to individual users. By declaring a line printer to be public, for example, an operator can ensure that all users have access to that commonly used output device.

4.4 SYSTEM MAINTENANCE FEATURES

4.4.1 Error Logging

RSX-11M provides an error logging facility as a system generation option for systems that are 24K words or larger. The error logging facility monitors the hardware reliability of an RSX-11M system; it continually detects and records information about hardware errors as they occur, regardless of whether the error is recoverable. Then, at user-determined intervals, a formatting task can be run to generate individual error and/or summary reports on some or all of these errors. The Executive automatically retries recoverable errors; whether or not recovery is successful, the user might be unaware that the error occurred unless the system includes the error logging facility.

In summary, error logging performs the following functions:

- Detects a hardware error as it occurs
- Gathers information about the error
- Stores the information in a file
- Formats the information to produce an error report

Control of the facility is shared between routines in the Executive and specific error logging tasks. These routines and tasks interface with each other to carry out the four operations described above.

An operator can generate a wide variety of error reports. Among many options, the user can specify a report that covers a certain time period, a certain device or group of devices, or perhaps a certain type of error. The user may also request a report that contains only information on individual errors, one that contains only summary information, or one that contains both kinds of statistics.

Because the error log files may be written to a removable volume, an operator can generate the reports either on site or at any other RSX-11M installation that supports the error logging facility.

4.4.2 Diagnostic Tasks

RSX-11M also provides a group of diagnostic tasks for which Executive support can be incorporated at system generation. A diagnostic task tests a specific device to identify the source of any errors. RSX-11M

diagnostic tasks test for malfunctions on most disks, DECtapes, magnetic tapes, and terminals. The tasks are simple to use and require little storage space.

When used in connection with error logging reports, the diagnostic tasks can significantly reduce system downtime. The system manager should regularly generate error reports to check on hardware performance. When a number of errors indicates that a particular device is beginning to malfunction, the manager can run the diagnostic task for the erring device to help isolate the source of the errors.

Each diagnostic task has two modes of operation: customer mode and service mode. In customer mode, the user simply activates the appropriate task, which then runs to completion and reports its findings. (Because the tests destroy any data resident on the device being tested, only authorized users should be allowed to run diagnostic tasks.) Service mode is intended for use by DIGITAL Field Service engineers. Service mode allows the user to modify the test content initially and to interrupt the running test to make further modifications.

4.4.3 Power Failure Restart

RSX-11M includes a power failure restart facility that smooths out intermittent short-term power fluctuations with little loss of service or data. The facility functions in four phases:

- When power begins to fail, the CPU traps to the Executive, where volatile register contents are stored, thereby gracefully bringing system operations to a halt.

- When power is restored, the Executive again receives control and restores the preserved state of the system.

- The Executive then schedules all device drivers that were active at the time of the power failure at their power-fail entry points (refer to Section 6.3 for a discussion of device drivers). Drivers have the option of being scheduled either:

  1. Whenever power fails, or

  2. Only when power fails while the driver is servicing an I/O request.

  The drivers can then make any necessary restorations of state (repeat an I/O transfer, for example).

- The Executive then determines if any user-level tasks have requested notification of power failure (issued a system directive that requested an AST on power recovery). The Executive then initiates AST's for any tasks that have requested them.

CHAPTER 5

PROGRAM DEVELOPMENT

This chapter discusses RSX-11M features that pertain to user program development.

5.1 PROGRAMMING LANGUAGES AND SORT UTILITY

RSX-11M supports several programming languages and a powerful sorting utility:

- MACRO-11
- FORTRAN IV and FORTRAN IV-PLUS
- COBOL
- SORT-11
- BASIC-11
- BASIC-PLUS-2

MACRO-11 is the standard language for RSX-11M; translators for the other supported languages are optional and must be purchased separately.

5.1.1 MACRO-11

The programmer working closely with the PDP-11 hardware can use the powerful MACRO-11 Assembler. In addition to allowing the user to write code using the machine-language instructions, MACRO-11 allows the programmer to define "macros" that may be invoked to generate repetitive coding sequences.

5.1.2 FORTRAN

The FORTRAN (FORmula TRANslation) language is especially useful in scientific and mathematical applications. PDP-11 FORTRAN conforms to the specifications of American National Standard FORTRAN (X3.9-1966), and includes substantial extensions to that standard.

The FORTRAN system consists of a compiler, a library of functions, and an object time system (OTS). RSX-11M also provides a set of FORTRAN callable Instrument Standard of America (ISA) process control subroutines. The compiler produces object code from the source

program, which is then linked by the Task Builder to create a task. The OTS consists of routines that are selectively linked with the user's program to perform certain arithmetic, I/O, and system-dependent service operations. The OTS also detects and reports runtime error conditions.

RSX-11M supports both FORTRAN IV and its superset, called FORTRAN IV-PLUS. FORTRAN IV-PLUS is an optimizing compiler that generates highly efficient code that utilizes the floating point processor.

5.1.3 COBOL

COBOL (COmmon Business Oriented Language) is an English-like programming language designed primarily for business use. PDP-11 COBOL conforms to the American National Standard 1974 COBOL standard, and contains many high-level features. Individual COBOL modules conform to the American National Standard implementation levels as follows:

Module	Level
Nucleus	2
Table handling	2
Sequential I/O	2
Relative I/O	2
Indexed I/O	2
Segmentation	2
Library	1 and part of 2
Interprogram Communication	1

5.1.4 SORT-11

The SORT-11 utility program accepts an input file from any peripheral device. Based on the sorting technique selected by the user, SORT-11 either sorts the contents of the input file or extracts and sorts key information from the records of the input record into a permanent output file.

5.1.5 BASIC-11

BASIC-11 is easy to learn and use, and has wide acceptance in educational, business, and scientific applications. PDP-11 BASIC's "immediate" mode allows each statement to be executed as it is typed in; thus the computer can be used like a desk calculator. Alternatively, a program can be entered, edited, and then run as a unit.

5.1.6 BASIC-PLUS-2

Unlike BASIC-11, BASIC-PLUS-2 is a compiler system that is designed to provide the enhanced speed of execution that is possible with a compiled task. Additionally, BASIC-PLUS-2 provides easily-used debugging features and a language integrated I/O syntax that gives the user convenient access to the file handling and data record manipulation facilities of RMS-11 (see Section 6.2.2).

5.2 EDITING UTILITIES

RSX-11M provides three utility tasks for creating and editing source files. These editors are:

- The DEC Editor (EDT)

  EDT is a standard text editor offered on most Digital Operating Systems. Thus, it represents a common editing language across systems. EDT's features include:

  - English language-based commands
  - Hard-copy and screen display
  - On-line error diagnosis
  - File and buffer I/O handling
  - Line and Character Editing modes
  - Maneuverable cursor and line-pointer

- The Line Text Editor Utility Program (EDI)

  EDI is an interactive context-editing program that provides the capability to create and modify source programs and other ASCII text material. EDI can be directed by means of terminal commands to read a line, or group of lines, from the input file into an internal buffer. The user can then, by means of additional commands, examine, delete, and change text, and insert new text at any point in the buffer. After the line or block of lines has been edited, the user can issue a command to write the data into a new file.

- The Source Language Input Program (SLP)

  SLP is a batch-oriented line editing program that is used to create and maintain source language files on disk. SLP also permits the user to create indirect files that contain SLP edit-control commands. The edit-control commands either delete, replace, or insert a line of text. SLP can also be used to obtain a line-numbered listing of a file as an editing aid.

5.3 DEBUGGING AIDS

5.3.1 Online Debugging Tool (ODT)

RSX-11M supports the Online Debugging Tool (ODT), an octal debugger, to help programmers debug programs written in MACRO-11. A programmer incorporates ODT at task build, and then interacts with ODT and the object program from an interactive terminal to:

- Print the contents of any location for examination or alteration,
- Run all or any portion of the object program using the breakpoint feature,

- Search the object program for specific bit patterns,
- Search the object program for words that reference a specific word,
- Calculate a block of words or bytes with a designated value, and
- Fill a block of words or bytes with a designated value.

The breakpoint is one of ODT's most useful features. It allows the user, when debugging a program, to run the program to a predetermined point, and at that point to examine and possibly modify the contents of various registers or locations. In this way, ODT acts as a monitor to the user program.

5.3.2 Post-Mortem and Snap-Shot Dumps

RSX-11M includes a task called PMD that produces "post-mortem" dumps after a task terminates abnormally and "snap-shot" dumps requested by programs. The dumps produced by PMD provide the following information:

- The task name and reason for the dump,
- The contents of the task's registers, stack, and program counter at the time of the dump,
- The devices being used and the files open at the time of the dump,
- The status bits, event flags, and I/O count for the task,
- The terminal that initiated the task,
- The device from which the task was loaded, and the physical address of the task image file, and
- The task's virtual memory, displayed in word octal, Radix-50, byte octal, and ASCII.

Programmers can use these dumps to debug a program offline, and when the program has not been linked to ODT.

5.4 THE TASK BUILDER

A MACRO or a higher level language programmer performs four basic steps to create a task and to prepare it for execution; the programmer must:

1. Write a source program and then enter it into a source file from a terminal,
2. Submit the source file to the MACRO-11 Assembler or to the appropriate higher level language compiler,
3. Create an executable task by submitting the object code (the assembled or compiled program) to the Task Builder, and
4. Issue an MCR command to install the task image (see Section 3.2.1).

Figure 5-1 illustrates the steps in creating a FORTRAN task.

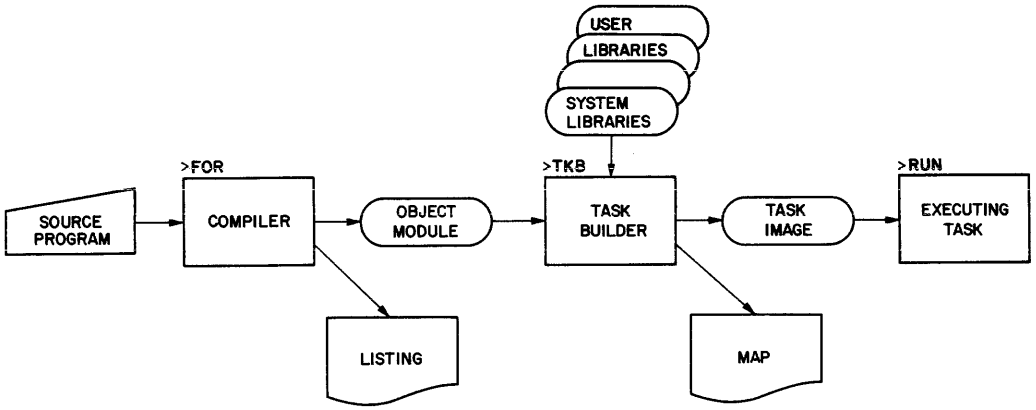

Figure 5-1 Steps in Creating a FORTRAN Task

Task Builder functions include:

- Linking object modules,
- Resolving any references to a system object-module library or user object-module library,
- Allocating required system and task memory,
- Producing an optional memory map file that describes the allocation of storage, the separate modules that comprise the task, and all global-symbol values,
- Building an overlaid task according to an overlay description supplied by the user.
- Linking the task to shared regions, which are blocks of code or data that are shared among numerous tasks.

CHAPTER 6

FILES AND I/O OPERATIONS

This chapter introduces the file system supported by RSX-11M and describes some basic aspects of I/O operations. In RSX-11M, a file is an owner-named area on a volume; a volume is a magnetic medium such as a disk, a DECtape, or a magnetic tape.

6.1 FILES-11

Files-11 is a software system that oversees the storage and handling of files held on volumes. Files-11 volumes are magnetic media that have been specially formatted by means of an MCR command called Initialize Volume. (Files-11 magnetic tapes conform to the American National Standard Magnetic Tape Labels and File Structure for Information Interchange X3.27-1969.) Volumes that are not properly formatted are considered to be "foreign." Files-11 cannot directly access foreign volumes; however, RSX-11M includes a file exchange utility that translates files in DIGITAL's DOS or RT-11 format into Files-11 format.

Files-11 is the standard file system for the RSX-11D and IAS operating systems, as well as for RSX-11M. Files-11 volumes are therefore completely compatible among all three operating systems.

User tasks can access foreign volumes to perform I/O independently of Files-11; this capability is often necessary for real-time applications.

6.1.1 File Ownership and Directories

When an RSX-11M user creates a file, the system places the file name in a User File Directory (UFD), and stores the user's current User Identification Code (UIC) with the file name to indicate the owner of the file. In most cases, the UFD corresponds to the owner UIC; but a file can be listed in a UFD that is not related to the owner code. It is also possible for a file to be listed simultaneously in more than one UFD.

A UFD is itself a file that a user must explicitly create by means of the MCR command UFD. A user specifies a UFD in the format of a UIC: [g,m] where g and m are octal numbers that represent the user's group and member number respectively. The actual name of the UFD is a concatenation of the group and member numbers, terminated by .DIR. For example, the name of the directory that corresponds to the UIC [203,165] is 203165.DIR. All UFDs are listed in each volume's Master File Directory (MFD), which corresponds to UIC [0,0] and is therefore named 000000.DIR. The directories (both UFDs and MFDs) contain the

names of files as well as pointers to each file's header. The file header contains information about the physical location of the file segments.

6.1.2 File Protection

Anyone who wants to access a file must know the UFD in which it is listed. This knowledge, however, is not sufficient to allow access. A user must also satisfy conditions specified in a protection mask associated with the file to be accessed.

The protection mask specifies the types of access allowed to four defined user groups. The four types of access are read, write, extend, and delete. The four user groups, defined according to UIC, are:

- System tasks and users that have a privileged UIC
- Owner tasks and users that run under the same UIC as the file's owner
- Tasks and users whose UIC is in the same group as the file's owner
- All other tasks and users

6.1.3 File Specifiers

To refer to a file, a user supplies a standard file specifier in an MCR or task command string. The specifier includes the following information:

- The device holding the volume that contains the file
- The UFD in which the file is listed
- The filename
- An abbreviation that describes the contents of the file (that is, the file "type")
- A number that differentiates one version of the same file from another

The file system uses the information supplied in the specifier to retrieve the file and to act upon it as directed by the command string in which the specifier appears.

6.1.4 File Manipulation

The Files-11 system disguises differences between files held on different types of volumes and ensures the integrity of files as they are transferred from one type of volume to another.

Before a file can be accessed, the volume that contains the file must be known to the system. The user issues an MCR command to perform this operation. Once the required volumes are ready, the user can manipulate files by means of file utilities or user-written tasks.

The most commonly used file utility is the Peripheral Interchange Program (PIP). The major functions performed by PIP are:

- Copying files from one device to another
- Deleting files
- Renaming files
- Listing file directories

6.2 TASK I/O OPERATIONS

6.2.1 File Control Services and Record Management Services

User tasks running on RSX-11M access data within files through the use of two sets of subroutines:

- File Control Services (FCS)
- Record Management Services (RMS)

FCS and RMS are to individual files what Files-11 software is to entire volumes. As discussed previously, Files-11 routines oversee the storage and handling of files on volumes. Through the software mechanisms of MFDs, UFDs, file specifiers, and protection classifications, Files-11 provides access to volumes, imposes a logical structure upon them, and maintains their integrity. FCS and RMS, on the other hand, oversee the storage and handling of data within files. They provide access to individual files, impose logical structures upon them, and maintain the integrity of the user data they contain.

Through FCS or RMS, user tasks access files on Files-11 volumes and process their contents using either block-oriented or record-oriented I/O operations. Block-oriented operations treat files as arrays of equal-sized structures called virtual blocks. The size of virtual blocks in a file is determined by the type of volume containing the file. On disk volumes, virtual blocks in all files are always 512 bytes long. On ANSI-compatible magnetic tapes, in contrast, the user has the option of specifying block sizes other than 512 bytes.

Record-oriented I/O operations treat files as collections of logical records. The size of individual logical records is specified by the user rather than being determined by the physical medium. FCS and RMS allow tasks to write records to files such that they can be subsequently retrieved at will.

The manner in which records can be retrieved depends on the logical structure, or file organization, imposed upon files by FCS or RMS. FCS imposes a single file organization upon all files. FCS views the contents of a file as a continuous sequence of user records. The sequence in which records appear is the same as the sequence in which they were written to the file. This logical structure is called the sequential file organization. Files on magnetic tape or card readers, for example, are of necessity sequentially organized since the physical media permit no other structure. However, FCS provides only a single file organization for all files regardless of medium.

For its sequentially organized files, FCS provides two methods (access modes) of storing and retrieving individual records. Sequential access mode stores successive records physically adjacent to previous records and retrieves records based on this adjacency. The second access mode -- random access -- is permitted only for disk files containing equal-sized records. In FCS random access mode, tasks can read and write records by specifying their relative position (from 1 to n) in the file. Successive I/O operations can access records anywhere within the file without regard to the inherent physical sequence.

In contrast to FCS, RMS provides three file organizations -- sequential, relative, and indexed (the indexed file organization is supported under separate license by the RMS-11K product). The sequential file organizations of FCS and RMS are identical. RMS, however, does not support FCS-style random access to sequential files. Rather, through the relative file organization, RMS allows records to be read and written either sequentially or by specifying their relative position (again a number from 1 to n) from the beginning of the file. Finally, the indexed file organization allows tasks to retrieve records randomly by specifying the contents of identifying fields (called key fields) within the records.

6.2.2 Device Independence

Using FCS and RMS, an RSX-11M programmer codes independently of the specific physical characteristics of devices; that is, the programmer does not need to know which devices the program eventually uses for its I/O operations (except for the requirement that random access mode can be performed only on disk files). A program performs I/O on Logical Unit Numbers (LUNs), which the programmer or an operator subsequently assigns to specific devices before the program actively uses the LUNs.

An operator can subsequently issue an MCR command to change the physical device used for I/O, if the device fails, for example. The command redirects all I/O intended for one device to another device of the same or similar type. The redirection has no effect on LUN assignments; and FCS or RMS makes the change in device characteristics transparent to the programmer.

Logical devices provide another means of associating LUNs with physical devices. Instead of directly assigning a LUN to a physical device, a task can assign a LUN to a logical device. An operator must then issue an MCR command to associate the logical device name with a physical device unit. A logical device name is similar to a logical unit number; it does not correspond to a physical device until a device is explicitly assigned to it.

6.3 PHYSICAL I/O OPERATIONS

Physical I/O operations involve the transmission of data between main memory and connected peripheral devices. In RSX-11M, a connected device is inoperable unless there is specific software resident in memory for that device type. This software can take the form of either an I/O driver or a user task that connects directly to a hardware vector associated with the device.

An I/O driver performs functions that enable physical I/O operations to occur. Drivers for most device types are built into the Executive

at system generation. RSX-11M includes drivers for all the standard supported devices. In addition, RSX-11M allows the incorporation of user-written I/O drivers for nonstandard devices. This capability is essential for real-time applications that communicate directly with devices such as production machinery or laboratory equipment.

Some installations have one or more devices that are only occasionally used. To avoid wasting memory with permanently resident but seldom used I/O drivers, a user can select a system generation option that allows drivers to be loaded and unloaded. Note, however, that some drivers are not loadable.

An alternative approach to I/O drivers is the connect to interrupt vector facility provided by the RSX-11M Executive. Through this facility, the user task itself, rather than an I/O driver, can receive the hardware interrupts generated through hardware vectors dedicated to the device.

INDEX

Acquisition,
  data, 2-1
Active task, 3-4
Address,
  physical, 3-2, 3-10
  virtual, 3-2, 3-10
Address space,
  extended, 3-9
  logical, 3-10
  physical, 3-10
  virtual, 3-10
Aids,
  debugging, 5-3
Allocating checkpoint space, 3-5
Allocation,
  device, 4-4
Applications,
  laboratory, 2-2
  medical, 2-2
  real-time, 2-1
  RSX-11M, 2-1
Assembler,
  MACRO-11, 5-1
AST, 3-9
AST entry points, 3-9
AST service routines, 3-9
Asynchronous System Traps, 3-9
Attached terminal, 4-4

BASIC, 5-2
BASIC-PLUS-2, 5-2
Basic MCR, 1-2
Batch editor, 5-3
Blocked task, 3-5
Breakpoint, 5-3
Builder,
  Task, 5-4

Central Processing Unit, 3-3
Checkpoint file, 3-6
Checkpoint space, 3-6
  allocating, 3-6
Checkpointing, 3-6
COBOL, 5-2
Code,
  User Identification, 4-4, 6-1
Command files,
  indirect, 4-2
Commands,
  MCR, 4-1
  multiuser, 4-3
  privileged, 4-2, 4-3
Communications,
  network, 2-3
Concurrent program execution, 3-3
Console Routine,
  Monitor, 4-1
Control,
  Executive, 3-3
Control Services,
  File, 6-3
CPU, 3-3
Creating a task, 5-4

Data acquisition, 2-1
Debugging aids, 5-3
Debugging Tool,
  Online, 5-3
DECNET-11, 2-3
DECNET-11M, 2-3
DEC Standard Editor (EDT), 5-3
Development,
  program, 2-1, 2-3
Device,
  logical, 6-4
Device allocation, 4-4
Device independence, 6-4
Devices,
  private, 4-4
  public, 4-4
Diagnostic tasks, 4-5
Directives,
  memory management, 3-10
  system, 3-9
Directory,
  Master File, 6-1
  System Task, 3-4
  User File, 6-1
Distribution of resources, 3-6
Dormant task, 3-4
Driver,
  I/O, 6-4
Dumps,
  post-mortem, 5-4
  snap-shot, 5-4

EDI, 5-3
Editor,
  batch, 5-3

Editors,
  text, 5-3
EDT, 5-3
Entry points,
  AST, 3-9
  SST, 3-9
Error logging, 4-5
Event flag numbers, 3-9
Event flags, 3-9
Events,
  real-time, 2-1
  significant, 3-7
Execution,
  concurrent program, 3-3
  sequential program, 3-3
Executive, 3-1
Executive control, 3-3
Extended address space, 3-9
External task scheduling, 4-2

Failure restart,
  power, 4-6
FCS, 6-3
File,
  checkpoint, 3-6
File Control Services, 6-3
File Directory,
  Master, 6-1
  User, 6-1
File exchange utility, 6-1
File manipulation, 6-2
File processor,
  indirect, 4-3
File protection, 6-2
File specifiers, 6-2
File symbols,
  indirect, 4-3
File system, 6-1
Files, 6-1
  indirect command, 4-2
Files-11 format, 6-1
Flag numbers,
  event, 3-9
Flags,
  event, 3-9
Foreign volumes, 6-1
Format,
  Files-11, 6-1
FORTRAN IV, 5-2
FORTRAN IV-PLUS, 5-2

Generation,
  system, 1-1

Identification Code,
  User, 4-4, 6-1
Image,
  task, 5-4
Indirect command files, 4-2
Indirect file processor, 4-3
Indirect file symbols, 4-3
Input,
  solicited, 4-4
Installation,
  task, 3-4
Installed task, 3-4
Interchange Program,
  Peripheral, 6-3
Interrupts,
  software, 3-9
I/O driver, 6-4
I/O operations, 6-1, 6-3

KT11 memory management unit,
    3-2, 3-10

Laboratory applications, 2-2
Languages,
  supported, 2-3, 5-1
Logging,
  error, 4-5
Logical address space, 3-10
Logical device, 6-4
Logical Unit Numbers, 6-4
LUN's, 6-4

MACRO-11 Assembler, 5-1
Maintenance,
  system, 4-5
Manipulation,
  file, 6-2
Manufacturing, 2-2
Mapped system, 3-1
Mask,
  protection, 6-2
Master File Directory, 6-1
MCR, 4-1
  Basic, 1-2
MCR commands, 4-1
Medical applications, 2-2
Memory, 3-1
Memory management
    directives, 3-10
Memory management unit,
  KT11, 3-2, 3-10

Memory organization, 3-1
Memory requirements,
  minimum, 1-1
MFD, 6-1
Minimum memory requirements, 1-1
Monitor Console Routine, 4-1
Multiprogramming,
  real-time, 3-1
Multiuser commands, 4-3
Multiuser protection system, 4-4

Network communications, 2-3
Numbers,
  event flag, 3-9
  Logical Unit, 6-4

ODT, 5-3
Online Debugging Tool, 5-3
Online Task Loader, 1-2
Operation,
  system, 4-1
  terminal, 4-3
Operations,
  I/O, 6-1, 6-3
Organization,
  memory, 3-1
OTL, 1-2

Partition, 3-1
  system-controlled, 3-2
  user-controlled, 3-2
Partition types, 3-2
Peripheral Interchange
    Program, 6-3
Physical address, 3-2, 3-10
Physical address space, 3-10
PIP, 6-3
PMD, 5-4
Points,
  AST entry, 3-9
  SST entry, 3-9
Post-mortem dumps, 5-4
Power failure restart, 4-6
Priority,
  swapping, 3-7
  task, 3-5
Private devices, 4-4
Privilege status,
  terminal, 4-3
Privileged commands, 4-2, 4-3
Process control, 2-2
Processor,
  indirect file, 4-3
Program development, 2-1, 2-3
Program execution,
  concurrent, 3-3
  sequential, 3-3
Programming languages,
  supported, 5-1
Programs, 3-1
Protection,
  file, 6-2
Protection mask, 6-2
Protection system,
  multiuser, 4-4
Public devices, 4-4

Ready-to-run task, 3-4
Real-time applications, 2-1
Real-time events, 2-1
Real-time multiprogramming, 3-1
Record Management Services (RMS),
    6-3
Regions,
  shared, 3-10
Requirements,
  minimum memory, 1-1
Resources,
  distribution of, 3-6
Restart,
  power failure, 4-6
Round-robin scheduling, 3-6
Routines,
  AST service, 3-9
  SST service, 3-9
RSX-11M applications, 2-1
RSX-11S, 1-2

Scheduling,
  external task, 4-2
  round-robin, 3-6
Sequential program
    execution, 3-3
Service routines,
  AST, 3-9
  SST, 3-9
Services,
  File Control, 6-3
  Record Management, 6-3
Shared regions, 3-10
Significant events, 3-7
Slave terminal, 4-4
SLP, 5-3
Snap-shot dumps, 5-4
Software interrupts, 3-9
Solicited input, 4-4
SORT-11, 5-2

Space,
  allocating checkpoint, 3-6
  checkpoint, 3-6
  extended address, 3-9
  logical address, 3-10
  physical address, 3-10
  virtual address, 3-10
Specifiers,
  file, 6-2
SST, 3-9
SST entry points, 3-9
SST service routines, 3-9
State,
  task, 3-4
Status,
  terminal privilege, 4-3
STD, 3-4
Subpartition, 3-2
Substitution,
  symbol value, 4-3
Supported languages, 5-1
Swapping, 3-6
Swapping priority, 3-6
Symbol value substitution, 4-3
Symbols,
  indirect file, 4-3
Synchronous System Traps, 3-9
System,
  file, 6-1
  mapped, 3-2
  multiuser protection, 4-4
  unmapped, 3-2
System directives, 3-9
System generation, 1-1
System maintenance, 4-5
System operation, 4-1
System Task Directory, 3-4
System Traps,
  Asynchronous, 3-9
  Synchronous, 3-9
System-controlled partition, 3-2

Task,
  active, 3-4
  blocked, 3-4
  creating a, 5-4
  dormant, 3-4
Task (Cont.),
  installed, 3-4
  ready-to-run, 3-4
Task Builder, 5-4
Task Directory,
  System, 3-4
Task image, 5-4
Task installation, 3-4
Task priority, 3-5
Task scheduling,
  external, 4-2
Task state, 3-5
Tasks, 3-1
  diagnostic, 4-5
Terminal,
  attached, 4-4
  slave, 4-4
Terminal operation, 4-3
Terminal privilege status, 4-3
Text editors, 5-3
Tool,
  Online Debugging, 5-3
Traps,
  Asynchronous System, 3-9
  Synchronous System, 3-9
  system, 3-9

UFD, 6-1
UIC, 4-4, 6-1
Unit Numbers,
  Logical, 6-4
Unmapped system, 3-1
User File Directory, 6-1
User Identification Code, 4-4, 6-1
User-controlled partition, 3-2
Utility,
  file exchange, 6-1

Value substitution,
  symbol, 4-3
Virtual address, 3-2, 3-10
Virtual address space, 3-10
Volumes, 6-1
  foreign, 6-1

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.