THE UNIVERSITY OF MICHIGAN

Memorandum

RAMP: A PDP-8 MULTIPROGRAMMING SYSTEM FOR REAL-TIME DEVICE CONTROL

David Mills

CONCOMP: Research in Conversational Use of Computers
F. H. Westervelt, Director
ORA Project 07449

supported by:

DEPARTMENT OF DEFENSE
ADVANCED RESEARCH PROJECTS AGENCY
WASHINGTON, D. C.

CONTRACT NO. DA-49-083 OSA-3050
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION, ANN ARBOR

May 1967

TABLE OF CONTENTS

RAMP: A PDP-8 MULTIPROGRAMMING SYSTEM FOR REAL-TIME DEVICE CONTROL

INTRODUCTION

The following sections describe the organization of a PDP-8 multiprogramming system that provides real-time device control and task-switching operations. The system, called RAMP* for convenience here, operates under interrupt control using integral task queue and buffer management routines. A simple command language is implemented, which provides control over both debugging and normal operational procedures and operates entirely within the system. The system has been adapted to a number of device-control applications including those of large-scale audio and data circuit switching, and computer-driven cathode ray display supervision.

The techniques used are believed to represent a good compromise between the large memory requirements of a full-scale general multiprogramming system and a more specialized system tailored for faster but more restricted operation. Some of these techniques may seem a little strange to conventional multiprogramming practice and for this reason are described in greater detail below.

GENERAL ORGANIZATION

The principal components of the basic RAMP system include:

1. An interrupt identifier that identifies the device causing an interrupt and calls the appropriate subroutine;

* Random Access Multiple Program, from the system by the same name for which this program was originally written.

2. a set of interrupt routines that services the device causing an interrupt and transmits data between the device and an attached buffer;

3. a set of buffer management routines that services the buffers attached to the keyboard, printer, and other special devices;

4. a task-switching processor that maintains a queue of active tasks and executes each as required;

5. a set of utility routines for keyboard and printer character formatting and conversion; and

6. a command language interpreter that decodes system commands entered via the teletype keyboard.

The basic RAMP system, described in Part I of this report, provides a nucleus around which dependent subsystems can be constructed. This organization is so arranged that additional operations can be entered in the system rather conveniently. A particularly useful operation involves data transmission between two or more devices attached to the PDP-8 or between two or more PDP-8 RAMP systems connected by some kind of data set. Part II of this report describes the extensions to the basic nucleus which make these operations possible.

PART I: THE BASIC SYSTEM NUCLEUS

INTERRUPT IDENTIFIER

When the PDP-8 interrupt system is enabled, a device request for service causes a forced JMS instruction to location 0 in memory. Following conventional practice, the source of the interrupt is identified by IOT-skips directed toward each device in turn, starting with that assigned the highest priority. When the device is identified, a JMS is entered to a subroutine that services the device and clears the interrupt signal at the device. When all such devices are serviced, control is returned to the interrupted program. Temporary storage for the Link and Accumulator contents is provided by the interrupt identifier routine, which also services the parity error and power failure/automatic re-start facilities.

INTERRUPT ROUTINES

Each interrupt routine typically services one device such as, for example, the keyboard or the printer. A class of devices with similar characteristics may be serviced with a single interrupt routine by providing each device with a block of control information that is accessed by all transmission routines serving the device. A table of such unit-control blocks (UCB's) may be constructed in such a way as to provide a computable mapping between the device number of the class requesting service and the location of its UCB entry in the table. In any case, the service requirements of the device must be satisfied by the interrupt routine before it returns control to the interrupt identifier. This includes transmitting its data, clearing its interrupt flag, and enabling it to receive or transmit any additional data. For the purposes of data transmission, a set of buffer-management routines is provided (see next section). Throughout the interrupt identifier and interrupt routines, the interrupt system must be disabled.

BUFFER MANAGEMENT ROUTINES

Each cyclic buffer region allocated for use by the system is associated with a five-word buffer control block (BCB) that contains the following information:

1. a pointer to information inbound to the buffer,
2. a pointer for information outbound from the buffer,
3. a flag used to determine whether the buffer is empty, full, or has "wrapped around,"
4. a pointer to the first word of the allocated buffer storage, and
5. a constant equal to the size of the storage allocated.

The BCB entries are automatically maintained by the buffer management routines and are not normally modified by the calling programs.

Transmission to and from a buffer is on a first-in first-out character-by-character basis. In order to conserve storage, the buffers are organized in a cyclic fashion; thus the characters stored are not necessarily in sequence in memory. Routines for storing (PUT), fetching (GET), and backspacing (BKSP - deletes last character stored) buffer entries are provided. Each of these routines expects a calling sequence with the character in the Accumulator at entry or return, as appropriate, and with a pointer to the appropriate BCB as an argument immediately following the calling JMS. Return to the calling sequence is either immediately following the argument or the next location following this, depending upon the overflow/underflow status of the buffer. Since these routines may be called by interrupt routines, all calling sequences must disable the interrupt system before entry.

<u>TASK-SWITCHING PROCESSOR</u>

Multiprogrammed operations are sustained by the TASK and INSERT routines in conjunction with the task queue, which itself is stored in a cyclic buffer. Each block of machine code which may operate asynchronously with other such blocks is termed a task and obeys certain interface criteria within the system. Although each task monopolizes processing facilities when it is active, an appropriate hierarchical organization built among its dependent daughter tasks allows execution of other tasks when the mother task must wait for some asynchronous event. The system behavior then appears to be one in which many sequentially executing tasks proceed simultaneously.

The manner in which this multiprogramming operation is done is controlled by an entity called the Task Control Block (TCB). Each TCB consists of three words:

1. a pointer to the entry point of a task to be executed,
2. a return pointer to the calling task,
3. an optional argument which may be used to transmit information between the calling and called tasks.

Each time the TASK or INSERT routine is called, a TCB is constructed of its arguments and placed in a buffer on a first-in first-out queue. The TASK routine, in addition to this operation, fetches the next TCB from this queue and prepares to enter the task identified by the entry-point pointer of the TCB. The INSERT routine may also be called by an interrupt routine, allowing tasks to be entered from this source.

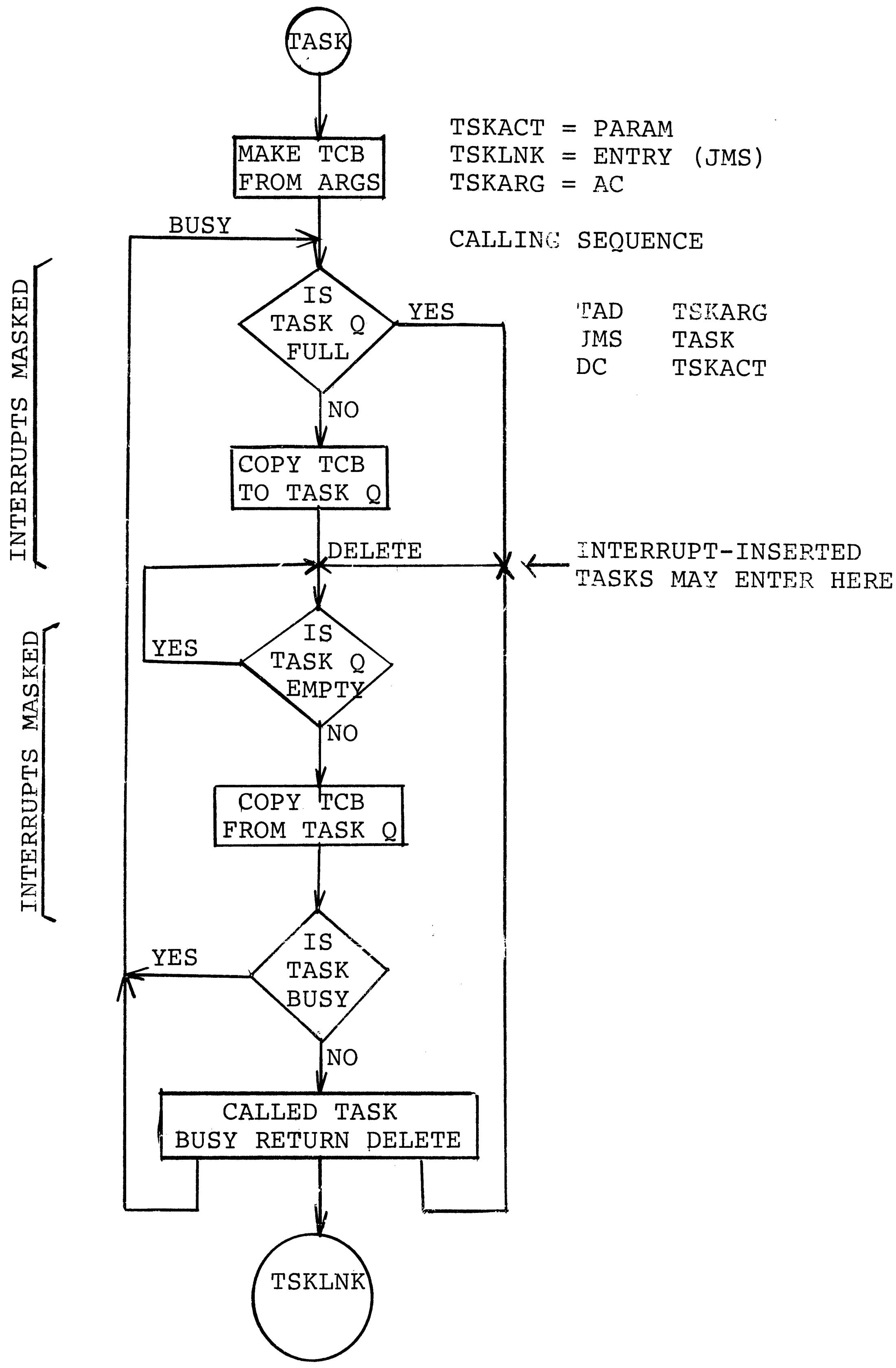
Each task itself is written very much like an ordinary subroutine, that is, it expects the return pointer to be stored as the first word at the entry point with execution beginning at the second word. When the task is entered from the task-switching processor, the TCB for the task becomes formally active and is stored in page zero for access by that task. Both the calling argument and the return pointer are available in the active TCB on entry. If a task is written so that it returns directly to the task-switching processor (see below) rather than to a calling task, then the return pointer may be used as a second calling argument. A return argument may be stored in the TCB on exit for use by the calling task.

A task may invoke a daughter task in either of two ways. If the task calls INSERT (with the interrupt system disabled), then a TCB for the daughter task is appended to the task queue and return is made immediately to the calling task. If the task calls TASK, then a TCB for the daughter task is appended to the task queue and the next TCB in the task queue becomes the active TCB. Return to the calling task is made only when the daughter task exits through its return pointer.

In order to avoid the use of re-entrant code (which is rather awkward to construct for a machine without an index register), where possible a convention is adopted that assigns a dynamic "re-enterability attribute" to every task in the system. If the entry point of a task is nonzero, then the task-switching processor will not allow a TCB pointing to this task to become active, but will immediately put the TCB at the end of the queue and fetch the next one. Thus if an active mother task sets up information prior to invoking a daughter task, that information is protected from being tampered with by a third task which seeks to enter the mother task while the daughter task is waiting on the queue. This interlock must be explicitly removed by the mother task before exit. In particular, if the mother task is indeed re-entrant, then the interlock can be removed by an appropriate DCA as the first executable instruction of the task. If the mother task may not be entered at any time during its active life, then a simple exit linkage can be used which saves the return pointer (stored at the entry point by the task-switching processor), stores a zero at the entry point, and then exits via the saved return pointer.

A task may exit in any of three ways. First, if during the execution of the task a temporary delay condition (such as a buffer overflow condition) exists which precludes its immediate progress, and furthermore if it is re-entrant up to this point, then it can exit through the return pointer BUSY. Such an action causes the task-switching processor to put the active TCB at the end of the queue and to fetch the next TCB. When the TCB for the task in question next appears as the active TCB, then the temporary delay condition can be tested and acted upon again. Second, if the task has no meaningful return pointer, and in particular if the TCB for the task has been created by an interrupt routine using the INSERT routine, then it can exit through the return pointer DELETE. Such an action simply deletes the active TCB and fetches the next TCB from the queue. Third, and in the general case, it may exit through the return pointer TSKLNK, which is one of the active TCB entries and is identical to the return pointer stored at the task-entry point by the task-switching processor upon entry to the active task. Such an action returns control to the mother task that invoked the active task.

The operation of the task-switching processor is shown in flow-chart form in Figure 1. The normal idling path, when no tasks are active or in the task queue, is the one-box loop beginning at DELETE. During this loop, the interrupt system is toggled so that a task may be entered on the queue by an interrupt routine using INSERT. The INSERT routine itself duplicates the operation of TASK from the entry point down to the level labeled DELETE. At this point INSERT returns to the calling program. Once a task has been entered on the task queue, the test immediately below DELETE falls through and the task is entered if possible. Note that if the task queue overflows, a TCB may be lost; and, furthermore, that the lost TCB may be due either to an INSERT or a TASK operation. In such a case, which should seldom if ever occur



```
TSKACT = PARAM
TSKLNK = ENTRY (JMS)
TSKARG = AC

CALLING SEQUENCE

     TAD     TSKARG
     JMS     TASK
     DC      TSKACT
```

Figure 1. Task Switching.

in a well-written system, certain non-re-entrant mother tasks may be locked out of the system for failure of a daughter task to return.

UTILITY ROUTINES

Certain often-used utility routines have been included within the system. These routines include those for single-character transmission between the processor and the attached keyboard/printer, as well as a set of formatting routines that read and print trimmed-ASCII strings and four-digit octal numbers. These routines are used by the debugging tasks and the command-language interpreter but may be used by other tasks embedded in the system as well.

Since the printer formatting routines are not re-entrant and since they call on each other, they must be reserved as a group by the task TPRSEL. A calling task thus first calls TPRSEL, at which point it is delayed until the line currently being printed is sent a carriage-return code. At this point an interlock is removed and any pending TPRSEL task in the task queue has the chance to set the interlock again and fall through to its mother task, which now has seized the printer routines and may call them via a simple JMS. Once the mother task calls upon the routine CRLF, the printer routines are automatically released for seizure by other pending TPRSEL tasks. This organization prevents lock-out of the system by a task that prints a large number of lines.

A corresponding interlock is placed in the keyboard conversion routines and is automatically maintained by the command language interpreter task CMMD (see below). This organization prevents a daughter task of CMMD reading more than one keyboard line at a time.

In addition to the keyboard and printer transmission and formatting routines, interrupt routines for these devices are included in the system. Since the use of the keyboard and printer is intended to be as a control and debugging console, the interrupt routines are tailored primarily for conversational operation and not for program loading or punching. In particular, inbound characters typed at the keyboard are checked for character-delete (back-arrow) and line-delete (rubout) codes and the indicated action taken. Furthermore, characters typed in are echoed or repeated to the operator on the printer. The echoing process is interlocked so that if the system is already printing a message on the printer, then the echo line typed at the keyboard is delayed until the printer has processed the next carriage-return code. This organization makes it feasible to "bang away" at the keyboard, no matter what is going on at the printer, and in particular to stop a runaway task that has seized the printer routines. Note that a line-feed code is generated automatically by the printer interrupt processor following a carriage return and therefore does not appear anywhere in the system except at this point.

An optional package of utility routines can be included in the system. This package contains the interrupt routines and buffering tasks for high-speed tape reader/punch equipment. These routines are multiprogrammed along with the rest of the system and may be used to transmit program or data tapes to and from the system. All character codes may be punched and all character codes except the XOFF code (ASCII 223) may be read from tape. The XOFF character is usually punched at the end of a program or data tape and is used to stop the reader.

The reader routines are provided with two options: in the first option, the reader routines will always attempt to keep the reader buffer full but will require the reader to fetch characters on a demand basis; in the second option, the reader will completely fill the reader buffer and will then wait until the buffer is empty before restarting to fill the buffer. The first option provides faster throughput for the reader routines but requires a PC01 reader; the second option allows the reader to operate in a burst-type block mode and is appropriate for use with older readers.

COMMAND-LANGUAGE INTERPRETER

The interface between the operator and the system is provided by the command-language interpreter task (CMMD). This task is inserted at every carriage-return detected in the keyboard text by the keyboard interrupt routine. The CMMD task decodes the command name entered by the operator and calls upon the various segments that decompose the remainder of the line and perform the actions required. If the action can be completed immediately, then the appropriate routines or tasks are called directly from the CMMD task. If the action involves an indefinite delay, such as dumping all of core or copying a tape, then the CMMD task inserts the appropriate task in the queue for later processing. In any case, each of the segments of CMMD exit in such a way that all unread text in the keyboard line is deleted before the CMMD task is itself declared not busy and exited.

Each command name consists of an indefinite number of characters, the first and last of which are looked up in an index that points to the appropriate segement of CMMD. Each of these segments may call upon the keyboard formatting routines for command parameters. In these routines, control characters (non-printing) are ignored; and, in general, non-alphanumeric codes are treated as break characters. Several such CMMD segments are included within the system, and others may be added easily. Those included deal with basic control and debugging functions and are summarized below:

DUMP (DP). Dump regions of memory in an octal format, eight words per line, prefixed by the location of the first word. If a single argument is given, then the word at the argument location is printed. If two arguments are given, then the block of memory between (and including) the argument locations is printed.

ALTER (AR). Store words in an octal format in memory. The storing operation is begun at the first argument location and continues for as many arguments as given, each argument being stored in ascending locations in memory. It is possible to modify any part of the system using the ALTER command; and, in particular, it is convenient to load short tapes of overrides via the keyboard tape reader.

RESET (RT). Immediately kills the system by drying up the task queue and voiding the keyboard and printer buffers. All outstanding tasks are "unbusied" by storing a zero at their entry points, using a dictionary table. It is possible to extend the list of buffers that are voided and to add to the reset dictionary to provide reset control for other tasks appended to the system.

TASK (TK). Insert (via INSERT) a task on the task queue. The first argument is the entry point of the task and the second is an optional argument to be passed to it, both in octal format. A task so invoked should exit only through the BUSY or DELETE return pointers, since the return pointer of a task inserted by the INSERT routine is not defined.

HELLO (HO). Prints a message as to the system maintenance level and version number.

HSR (HR). Start high-speed tape reader. The reader will fetch characters as required until an XOFF character is read. This command requires both the appropriate hardware and the service package described above under Utility Routines.

Other segments, when defined for the purpose of device control and task initiation, are expected to possess unique command names. If an unrecognizable command is decoded by the CMMD task, a diagnostic message is printed and the keyboard line deleted.

APPLICATIONS

From the nucleus of the basic system described, several complete systems for specialized real-time applications have been constructed. In one of these, which in fact inspired the construction of the system itself, the environment includes a number of electronic switches used to connect audio program sources to various listening positions where the listeners have real-time control over the selection of the program source (tape recorder) and the manner in which the source operates. The system is used to queue input information transmitted by each listener's console and to supervise the operation of the program source. In addition, each listener may record short utterances on special tape recorders that are seized on a demand basis by the system and attached to a particular listener's console. Each listener console and each tape recorder is attached as a separate device to a special scanning unit that helps identify the particular device requesting service. Up to 256 such devices are planned for this system, which currently includes 31, and each device may request service at up to ten times per second.

In a second application, the system has been used in a developmental processor for use in message switching, data formatting, and device control for remote terminal devices to be attached to a large time-shared computing system. A high-speed interface between the parent computer (IBM System/360 Model 67) and the PDP-8 has been constructed along with a special set of interfaces to a variety of data transmission equipment, including Western Electric 103, 201, 403, and 801 data sets operating at rates to 2000 bits per second. The system translates the various character sets used on remote terminals serviced by these data sets to the System/360 internal character codes, formats the message lines when appropriate, computes and checks message parity, and supervises the attachment of the various dial-up telephone lines connected to the data sets. Throughput in this application is at average rates to 10,000 bits per second and peak rates to 500,000 bits per second. Up to 64 data sets may be attached, each of which services a bidirectional transmission circuit, together with a high-speed paper tape reader/punch.

PART II:
BASIC SYSTEM EXTENSIONS FOR DEVICE CONTROL

The basic system nucleus described in Part I of this report includes support for the on-line teletype console and an optional high-speed paper tape reader/punch. The extensions to the basic system described below provide a mechanism for device and file maintenance and for data transmission between these devices and files and between the PDP-8 RAMP system and other machines using the appropriate data set transmission circuits. Transmission between the PDP-8 RAMP system and other machines is appropriate at any of several transmission rates using interface equipment specially designed for this purpose and described separately. The discussion below summarizes:

1. the device interface,
2. file structure, and
3. command extensions.

DEVICE INTERFACE

All input/output devices available to the RAMP system are identified by a unique logical device number (LDN). The devices in the system are assigned a contiguous set of these numbers starting at zero, so that an input device is assigned an even number and the most closely associated output device is assigned the subsequent odd number. Thus the on line keyboard may be assigned LDN 20 and the on-line teleprinter LDN 21.

Any task calling for input from or output to a device references one of these LDN's as appropriate. Only one task may attach an LDN at any time, during which the device is busy to all other tasks. Some tasks attach and LDN for the duration of a print line and release it at the end-of-line (carriage-return) code, while others attach an LDN for the duration of an entire file and release it only at an end-of-file code. Any task attempting to attach a busy input device is immediately aborted with an appropriate comment. Any task that attempts to attach a busy output device is placed in a wait condition pending release of the device by its supervisory task.

Associated with each LDN is a two-word entry in the Device Control Block (DCB) Table. These DCB entries, ordered by their LDN's, contain as the first word a pointer to a task that transmits a single character to or from the device as appropriate, and as the second word a switch. If this switch is zero, the device is available; if the switch is nonzero, the device is busy. The attaching task may use this switch for storage of temporary information, in particular LDN's of other attached devices, provided of course the values are nonzero.

By convention, each active and dormant task in the system has associated with it a source LDN and a sink LDN. These are packed in a single word (TSKDCB) which is part of the TCB for the task. This word, called the active DCB, is created when a task is inserted (by calling INSERT) and is propagated to all its daughter tasks. Any input/output operations specified by a task reference the active DCB for the task. The individual tasks are then responsible for the allocation of the devices themselves.

FILE STRUCTURES

All command/copy file operations reference a logical file as a unit and involve character transmission from one device to another. The source and sink involved are identified by the active DCB when a command/copy file operation is initiated by a task. The DCB for the initiating task, on the other hand, is called the master DCB. After initiating a command/copy operation, the initiating task stores the master DCB in the DCB table entry associated with the source device for the command/copy operation, thus reserving the source device for the duration of the operation. When a logical end-of-file is read from a source device, that device is released and a comment is written on the master sink device. The end-of-file indication is not transmitted to the sink device by the command/copy operation.

Two types of files may be distinguished: the copy file and the command file. The copy file consists of an arbitrary stream of characters that are transmitted from the active source device to the active sink device. The file is ended by an end-of-file character, which of course may not be part of any transmitted file. The command file consists of a stream of command lines to the command-language interpreter separated by carriage-return codes and ended by the END command. In this case, commands are interpreted from the active source device, and any output generated is transmitted to the active sink device.

Both the copy and command file operations operate asynchronously in a multiprogrammed fashion. However, separate copy operations may proceed in parallel for any number of devices in the system, while the command operation must proceed for only one source device at a time. Thus if a copy operation for a particular pair of devices must be delayed for a transmission operation, this delay does not affect copy operations outstanding for other pairs of devices. On the other hand, if a command operation must be delayed for a transmission operation, for example for a sink device to become available, then the entire command language interpreter is interlocked for this delay.

A special file behavior is specified for certain types of devices such as the online keyboard. Such devices operate slowly or at manual rates and may remain idle for some time until a command line is ended. On the other hand, such devices may transmit files of an indefinite length which would overflow any fixed-size buffer in the system. Accordingly, a dual behavior is specified for such devices: a latent state, in which the device buffers characters in real time until a carriage-return code is read; and the other, an active state in which characters are read from the receive buffer immediately as they become available. The latent state is typical of command operations in which the keyboard is used intermittently and is characterized by a low idling demand on the system processing resources. The active state is typical of both copy and command operations in which the active source device runs continuously, and is characterized by a moderate idling demand, but a faster response time.

The selection of the state under which a particular device is currently operating is made by the copy/command file routines. When either a copy or command file operation is initiated involving a particular active source device, then that device is place in the active state. Once a logical end-of-file is transmitted from a source device, that device reverts to the latent state.

When devices such as the online keyboard are in the latent state, characters transmitted from the device are stored in a dedicated buffer in real time. Line-editing functions such as character and line delete are performed in this real-time operation. When a carriage-return code is read from the source device, a command language interpreter task is inserted in the task queue, which in turn causes the CLI to read the edited buffer via the appropriate daughter task.

If a command file operation is outstanding against a particular source device, then that device is in the active state. Under these conditions the appropriate CLI daughter task reads characters from the source device buffer as they become available in the buffer, and no line-editing operation is possible.

COMMAND EXTENSIONS

Five commands are added to the basic system nucleus with the addition of the command/copy operations. These are ECHO, COPY, EOF, COMMAND, and END. Some of these commands may have logical device numbers as arguments. The devices for which service is provided in the basic system include the online keyboard and teleprinter, the high-speed paper tape reader and punch, an internal utility file, and an internal dummy file. Service routines for the online keyboard and teleprinter and high-speed paper tape reader and punch are described in Part I. Those for the two internal files are discussed immediately below.

The internal utility file consists of a first-in first-out buffer in core memory and a set of read-write tasks that interface with the system in the same manner as an input/output device. Information may be written into or read from the file by any operation that specifies the appropriate LDN. Overflows and underflows of the buffer will interlock the calling operation until the condition is cleared by another operation that transmits to or from the file as appropriate. The dummy internal file is intended as a "wastebasket" for garbage files. It serves as an infinite sink for write operations and as an infinite source of end-of-file characters for read operations.

Descriptions of the system commands follow in an Appendix. Note that the copy/command extensions to the basic system involve certain implications to the basic commands outlined in Part I. In particular, the DISPLAY and TASK commands will produce output on the master sink device as specified in the DCB supplied when the CLI task was created. In the case of the online keyboard operating in the latent state as the master source, the master sink is assigned as the online teleprinter. No other devices serviced by the basic system are operated in the latent state.

PART III: MECHANICS OF RAMP

The current version of the basic system nucleus of RAMP is assigned Version 12, while that including the copy/command operations, the high-speed papertape reader and punch, and the internal files is assigned Version 14. Operation and implementation information below is in reference to these versions, which will be referred to as simply 12 and 14 respectively.

Both 12 and 14 exist physically in three forms:

1. as symbolic source cards for 8SS, an assembler/simulater system for the PDP-8 and PDP-8S machines that is resident in the 7090 UMES operating system at the University of Michigan.
2. as binary object cards produced by 8SS; and
3. as a BIN object tape suitable for loading in the PDP-8 using the standard routines.

The source and object cards are handled using conventional batch-processing procedures by the IBM 7090 - System 360 Model 67 complex. The transcription of the object cards to punched tape is handled using the MTS system on the 360/67 and the facilities of the attached Data Concentrator, itself using a version of RAMP. Since both 12 and 14 run to well over 1000 source cards, the fast bulk I/O facilities of the 360/67 make this somewhat roundabout approach much more convenient than assembly using conventional DEC systems.

Both 12 and 14 have been carefully written to avoid use of special features of the 8SS assembler, notably use of macros and literals. A 7090-resident program has been written which, in conjunction with the 360/67 and the Data Concentrator, can be used to transcribe 8SS source cards to ASCII punched tape suitable for PAL-III processing. It is not recommended to process such a tape using slow-speed teletype input/output equipment. (It would take about four hours to make one pass over the source tape of some 150,000 characters!)

The program, once loaded, is started at location zero. A program loop through the task-scheduling routines about locations 400-600 is the normal idling path for the system. (A convenient thermometer for system environmental loading is the variable TSKCNT, which contains the number of dormant tasks on the task queue and may be displayed in a conveniently indicated register such as the MQ or a device register).

Commands may be entered in the system at any time from the keyboard. A line may be erased before it is entered by pressing the RUBOUT key, and one or more characters may be erased by pressing the ← key the appropriate number of times. The character codes for both of these operations can be changed by assembly parameters.

Once entered in the system, a command proceeds to completion. The system may be "dried up" at any time, however, by use of the RESET command. The command language interpreter must of course be free to accept this command. In any case, if the CLI is interlocked, starting the program at location CMDRSX will accomplish the same function as the RESET command. (Following this procedure, the RUBOUT key should be pressed to clear the keyboard buffer.) Input/output devices will complete real-time operations such as emptying output buffers, but input buffers and device assignments will remain unchanged after the RESET has become effective. For this reason, it may be necessary to store zeros in the DCB table (starting at DCBTBL) entries associated with the active devices using the manual controls of the machine. This procedure is necessary only when devices malfunction or a conflicting command/copy device assignment has been made.

Both 12 and 14 include service for both automatic restart and parity check options on the PDP-8. The automatic restart option is particularly useful in data transmission circuits where circuit configuration and device assignment are difficult to reestablish after a power failure due, say, to seasonal thunderstorms. The parity check option is standard on the PDP-8S; when included in a RAMP version the operation definition for SMP (skip on no memory parity error) is changed from 7401(unconditional skip) to 6101. No change is necessary in any case to the automatic restart code.

The basic nucleus occupies five pages in addition to about half of page zero. About three additional pages are necessary for the copy/command file routines, which include the service routines for the high-speed reader. Both of these figures include single-line buffers for the keyboard teleprinter, high-speed tape reader, and high-speed tape punch. The task-switching time on a PDP-8 with a single dormant task in the task queue is about 350 μs. This figure represents the setup time for any task in the system, whether it is in fact entered or not. It also represents the approximate time to enter a new task in the system (using INSERT) or to call a daughter task (using TASK).

APPENDIX

COPY MODULE COMMANDS FOR RAMP

DUMP (DP). Dump regions of memory in an octal format, eight words per line, prefixed by the location of the first word. If a single argument is given, then the word at the argument location is printed. If two arguments are given, then the block of memory between (and including) the argument locations is printed.

ALTER (AR). Store words in an octal format in memory. The storing operation is begun at the first argument location and continues for as many arguments as given, each argument being stored in ascending locations in memory. It is possible to modify any part of the system using the ALTER command; and, in particular, it is convenient to load short tapes of overrides via the keyboard tape reader.

RESET (RT). Immediately kills the system by drying up the task queue and voiding the keyboard and printer buffers. All outstanding tasks are "unbusied" by storing a zero at their entry points, using a dictionary table. It is possible to extend the list of buffers that are voided and to add to the reset dictionary to provide reset control for other tasks appended to the system.

TASK (TK). Insert (via INSERT) a task on the task queue. The first argument is the entry point of the task and the second is an optional argument to be passed to it, both in octal format. A task so invoked should exit only through the BUSY or DELETE return pointers, since the return pointer of a task inserted by the INSERT routine is not defined.

HELLO (HO). Prints a message as to the system maintenance level and version number.

HSR(HR).	Start high-speed tape reader. The reader will fetch characters as required until an XOFF character is read. This command requires both the appropriate hardware and the service package described above under Utility Routines.
ECHO.	Transmit all characters on this line after the single argument to the sink device identified as the argument.
EOF.	Transmit an EOF character to the sink device identified as the argument.
COPY.	Copy characters from the source device identified as the first argument to the sink device identified as the second argument until an EOF character is read from the source device. The EOF character is not transmitted to the sink device.
COMMAND.	Interpret characters read from the source device identified as the first argument as a command file to the command language interpreter. Output generated will be transmitted to the sink device identified as the second argument. The command file is terminated by an 'END' command.