

**stichting
mathematisch
centrum**



AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 75/77

FEBRUARI

R.VAN VLIET & W.WAKKER

MULTICORE 8
A TIMESHARING SYSTEM FOR THE PDP8 SERIES

2e boerhaavestraat 49 amsterdam

BIBLIOTHEEK MATHEMATISCH CENTRUM
—AMSTERDAM—

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

Multicore 8

A timesharing system for the PDP8 series.

by

R.van vliet & W.Wakker

Abstract

In this document we present the first version of the M C 8 (MULTICORE 8) system.

MC8 is an operating system for the D.E.C. PDP-8 computer. It offers the possibilities of multiprogramming (parallel execution of tasks) and timesharing.

Most of the concepts used have already been implemented in other operating systems for the PDP-8. The main concept we added is that of a virtual memory: 256k virtual core in 16-32k actual core. Due to this, the system is particularly powerful for those who implement large programs or programs requiring large buffers. The responsibility of swapping is then automatically deferred to the MC8 monitor.

In developing the system objectives like modularity, well-defined datastructures and clearness of flow of control competed with the need of writing a compact and fast system for a small machine. Much attention was paid to a strict intertask and task/monitor communication.

KEYWORDS & PHRASES: Real time operating System, peripheral processor, multi programming

CONTENTS

	Abstract
	Conventions and abbreviations
0	Introduction
1	The MC8 system
1.1	Tasks
1.1.1	Relocatable code
1.1.2	The static task list
1.1.3	The task controlblock
1.2	The intertask communication
1.2.1	Claiming tasks
1.3	The scheduler
1.3.1	The routines OURUNQ and INRUNQ
1.3.2	The monitor commands STOP and RESUME
1.3.3	The runchecks
1.4	The virtual core
1.4.1	paging algorithm
1.4.2	Datafield and instructionfield
1.4.3	Fieldtable and coremap
1.4.4	Locking and unlocking fields
1.4.5	GETFLD and FRFLD
1.4.6	Swapping
1.5	The interrupt section
1.5.1	The MC8 interrupt section
1.5.2	The skipchain
1.5.3	Intslots and related monitor commands
1.5.4	Connected routines
1.5.5	Changing the state of an intslot
1.6	Flow of control
1.6.1	Execution states
1.6.2	Task switching
1.6.3	Critical sections, scheduling
1.7	The general page 0
1.7.1	Storage of task registers
1.7.2	The one-task-only part
1.7.3	General constants
1.7.4	Pseudo instructions
1.7.5	Entrance of the monitor command section
1.7.6	Saving the pc
1.7.7	Emergency escapes from pseudo instructions
1.7.8	Storing and updating the general page 0
1.8	Storage allocation
1.8.1	Free core in field 0

1.8.2	Requesting and returning fields
1.8.3	The free core chain
1.8.4	The REQ PAG and RTN PAG commands
1.9	Monitor tasks
1.9.1	The idletask
1.9.2	The system diskdriver
1.9.3	The swaptask
1.9.4	The taskfetcher
1.9.5	The timer
1.9.6	The timeoutqueue
1.10	Monitor commands
2	Miscellaneous remarks
2.1	Creating tasks
2.2	Reentrant tasks
2.3	Protect tasks
2.4	task library
2.4.1	Families of tasks
2.5	Configurating MC8
2.6	Error handling
2.7	The system bulletin
3	Literature
App. A	Listing of the MC8 system
App. B	Two tasks for the MC8 system
App. C	The taskloader task
App. D	A system bulletin
App. E	List of monitor commands

Conventions and abbreviations

Throughout this document a number of conventions and abbreviations is used. We list them below.

Conventions.

1. Arrays are indexed from 0 upward. An entry is denoted as NAME[I], where NAME indicates the array and I the index.
2. Valid pointers never point at 0000. This value thus may be used as nil. It indicates the end of queues, free entries etc.
3. All names in this document that correspond to symbolic names in the listing of MC8 are denoted in capitals.
4. In the description of monitor commands and pseudo instructions changes in registers are indicated with a "becomes"-sign (:=).

Abbreviations.

abr	meaning	section	description
ac	accumulator		12-bit processor register.
arg1	argument 1		first argument of monitor command.
arg2	argument 2		second argument of monitor command.
config	configuration		
dfr	datafield register		3-bit processor register. points at the current datafield.
fcchain	free core chain	1.8.3	list of free core junks used by the page allocator.
fldtab	fieldtable	1.4.3	table holding information on virtual fields.
ifr	instructionfield register		3-bit processor register. points at the current instructionfield.
intreq	interrupt request	1.5	
intslot	interrupt slot	1.5.2	5 consecutive locations in the skipchain corresponding to one peripheral device.
mq	multiplier quotient		12-bit processor register.
msptr	message pointer	1.2	pointer to word2 (first information word) of a message.
pc	program counter		12-bit processor register. points at the next program instruction.

stl	static task list	1.1.2	table holding the keys to all tasks in the system.
st#	static task number	1.1.2	index of task's key in the stl, identifies task.
swreq	swap request	1.4.6	request to swap in a virtual field.
tcb	task controlblock	1.1.3	blocklet of core, holding information on tasks whose state is not initial.
tcbptr	task controlblock pointer	1.1.3	pointer to tcb[5].
toq	timeoutqueue	1.9.6	

0. Introduction

When near the end of 1975 we started at the design of a text manipulation sytem for the D.E.C. PDP-8 computer (small but cheap) *1, we were faced with the problem of choosing an operating system to base our system on. We plan a system [1] to which about 8 terminals can be connected. These terminals are to be used to type in or edit a text, or to control a more complicated text manipulation program, such as a formatter, a neatener, an indexing program or the like, that works on an interactive base. So we needed an operating system able to execute several fairly large programs in paralel *2 and in which a growing library of utility tasks could be made available to each user.

The system that best met these requirements was MULTI8 [2]. Its main restrictions were:

- A. Only a limited number of tasks (about 64) could be kept in its task library. Moreover, the support of each terminal in MULTI8 took 5 tasks.
- B. The total amount of core in use by active tasks cannot exceed the actual coresize. it is nice to get around this restriction, as in fact tasks are executed one after another. The restriction may lead to cumbersome swapping of program overlays and buffers. Remedying these restrictions seemed quite complicated. Therefore we chose the challenging solution of writing our own operating system. MULTICORE 8 (abbreviated to MC8) is the result.

In developing MC8 great advantage was taken from our knowledge of the systems MULTI8 [2] and RTS8 [3]. A number of concepts was copied without change. Important new concepts are:

- A. The possibility of adding and removing families of tasks at run time.

The MC8 system can handle only 128 tasks at a time, but these tasks may be drawn from a large task library on disk. In the current task library 21 families of upto 31 tasks each may be stored.

*1 For those not familiar with the PDP-8 the most important features are listed in [7].

*2 The PDP-8 cannot actually execute tasks in paralel. By paralel we mean that tasks are worked upon for a while, one after another, so that to the terminal user it seems to occur simultaneously.

- B. The implementation of a virtual memory.
The total amount of core that may be accessed by active tasks is 256k. Each time a field (4k) of this core is actually accessed, the system swaps that field in one of the actual corefields.
- C. The highly formalised intertask communication.
This enables tasks to wait for multiple conditions. I.e., a task may set itself to wait for several conditions at a time. It will be continued as soon as one of these conditions is fulfilled. For example, a task may send messages to other tasks and subsequently wait for the first message that is reported.

MC8 can be run on a PDP-8/I, PDP-8/E or PDP-8/A computer. It requires at least 16k core, a fast background storage and a clock. The coresident portion of the system (called the monitor) occupies about 2k core memory allocated in field 0. The rest of this field is used by the avallist system. In other fields taskcode and buffers are allocated.

In appendix A a first version of the MC8 monitor is listed. It includes the interrupt section, the scheduler, the storage allocator, the virtual core manager, the monitor command section, the avallist system and a number of so-called monitor tasks, such as the diskdriver, the timer, the swaptask, the taskfetcher and the idletask.

Upon this groundlevel, the monitor of MC8, a higher level must be built containing a file system, a terminal section, device drivers etc. This document only deals with the MC8 monitor, which is to be a flexible base for more complicated systems.

The smallness of the PDP-8 is a serious constraint. To some extent this is reflected in a loss of modularity and locality. Sometimes this constraint forced the use of bad or slow algorithms in favour of better or faster ones that require more core (e.g., the swap strategy for the virtual core). The most eyestrking drawbacks of the system are the lack of intertask protection and the possibility of uncontrolled resource consumption.

Our first experience with the performance of the system is encouraging, as may be seen from the system bulletin in appendix D.

Chapter 1 discusses the important features of the MC8 system. In chapter 2 a number of miscellaneous remarks will be made on the performance of the system and how to use it. The monitor as a whole is listed in appendix A and some examples of tasks are given in appendices C and D.

1. The MC8 system

Chapter 1 is not intended to give a precise description of the MC8 system. Much more it is the intention to explain the algorithms used and give some insight in the ideas involved.

1.1 tasks

The basic unit in the MC8 system is a task. The monitor of MC8 constitutes an environment in which tasks may be run. It determines the order in which tasks are run, it provides means for intertask communication and for communication between a task and a peripheral, it hands buffers to tasks and performs a lot of other useful functions.

Tasks in the MC8 system belong to one of three categories:

Monitor tasks.

These tasks are effectively a part of the monitor. Their code is built in the monitor and closely interrelated with other parts of the system. They are discussed in sec. 1.9.

System tasks.

These tasks constitute the multiprogramming level of the MC8 system. The rest of this section is mainly concerned with system tasks. In general they are written and assembled as separate modules, knowing little about the monitor and in principle nothing about other tasks. Of course one could implement a family of tasks communicating via a well-defined interface.

Protect tasks.

These tasks constitute the MC8 timesharing level. They have no access to any other part of the system and run with the hardware usermode flipflop set. In this paper little is said on this category of tasks, as we did not decide yet on the exact way of implementing these tasks (see however sec. 2.3).

1.1.1 relocatable code

A task is a piece of executable code. When it is loaded into the system, a status is attached to it. The code of a task covers 1-31 consecutive pages of core *1. During execution of a task, the code is stored in consecutive pages of one memoryfield, never in page 0 of a field. Once the code has been stored in the virtual core, it is left there until the task itself specifies a SWAPOUT or EXIT command to

*1 for an exception see sec. 2.3.

the monitor.

At assembly time it is uncertain where in a memoryfield the code of a task will be stored during execution. This complicates off-page referencing *2.

The following relocation conventions are used (compare to MULTI8 [2]):

- A. The task is assembled as if its first page were stored in page 1 (loc0200-0377 oct) of a memoryfield.
- B. Each page of code starts with a, possibly empty, sequence of (off-page) pointers. The sequence is terminated by a location pointing at 0000. The latter cannot be confused with an actual pointer, because -as a consequence of A- 0000 does not belong to the taskcode.
- C. Before actually storing the taskcode in core, the contents of these first locations of each page are updated by subtracting 0200 oct and adding the actual first address of the code.

The PDP-8 assembler PAL8 [4] has an option to generate pointers at the end of a page. Writing " (name ", where an operand is allowed, generates such a pointer to the location "name". We created a special version of the assembler, in which pointers at the beginning of a page are generated by writing ")name ", where an operand is allowed [5].

1.1.2 The static task list (stl)

To each task that is loaded in the system an entry in the "static task list" (stl) is assigned. The index of this entry is called the "static tasknumber" (st#) and identifies the task in the system. The number of entries in the stl (i.e., the maximum number of tasks that may be loaded in the system at a given instant of time) is determined by the config parameter MAXSTL. MAXSTL must be in the range $0 \leq \text{MAXSTL} \leq 127$.

Free entries (=entries not corresponding to a task) in the stl are marked by denoting 0000 in their first word.

Each entry consists of 2 words:

word bit meaning

0 If<0 pointer to task controlblock,
 If=0 free entry,
 If>0 blocknumber of taskcode on disk.

1 0-2 Task's priority. $0 < \text{priority} \leq 7$. Used by the scheduler (see sec. 1.3).

*2 On-page referencing does not cause troubles, as these references are coded relative to the page-offset.

- 1 3 Zerorequestbit. If set, that task needs the
 "one-task-only" part of page 0 of the field in which it is
 run (see sec. 1.7.2).
- 1 5 Coreresidentbit. If set, the task must run in a
 coreresident field (see sec. 1.4.5).
- 1 7-11 Length of taskcode in pages.

The contents of word1 and the blocknumber of the taskcode on disk may be found in the directory of the task library.

Its entry in the stl is a key to all information on a task in the system. This may be seen from the following scheme of operation. When a task is loaded, first a free entry in the stl is searched, designating the st#. The entry is filled from the directory of the task library. In word0 the blocknumber of taskcode on disk is denoted. Blocknumbers must be in the range $1 \leq \text{blocknumber} \leq 2047$, that is a positive integer in the PDP-8. From the fact that word0 is >0 the system concludes that the taskcode is still on disk and the status of the task is initial. The latter implies that the task is not runnable, but waiting for a message. As soon as a message is sent to the task, a task controlblock (tcb) is requested. In the tcb the status of the task will be kept. The system prefills the tcb with initial values (see sec. 1.1.3). A pointer to the tcb is stored in word0 of the entry of the stl. A tcb is a blocklet of core, allocated in field 0, the monitor field, at addresses ≥ 2048 . Consequently the contents of word0 is now negative. This is the common situation. Normal message-receive procedures can now be executed.

The use of a positive blocknumber in word0 must be viewed as pure optimisation. Instead of denoting the blocknumber in word0, we might have attached a tcb to the task immediately when it was loaded. The blocknumber is denoted in tcb[15]. Many tasks are loaded into the system just to be addressable for other tasks. They provide functions that are rarely used, such as error recovery. Attaching a tcb to such tasks would cost a lot of scarce core in field 0. A positive value in word0 is perfectly equivalent to the situation that a tcb holding initial values is attached to the task.

When a task has completed its function it executes the EXIT command. This indicates that the task returns to its initial situation. Its code is erased from core and its tcb is returned. The disk blocknumber is rewritten in word0 of the entry in the stl.

A task is unloaded as follows:

First the task is reset to the initial situation. The only rudiment of the task in the system is now its entry in the stl.

Next this entry is marked as free by clearing word0. The task loses its st#, which makes it unidentifiable. It vanished from the system.

1.1.3 The task controlblock (tcb)

A task controlblock (tcb) is a blocklet of 16 consecutive locations of core, allocated in field 0. When the status of a task is not initial, a tcb is attached to it. During execution the status of a task is loaded into the hardware registers. It is restored into the tcb when the task is interrupted or waiting for some event. The tcb also holds information for the claim mechanism, the intertask communication, the scheduler and the taskfetcher.

Layout of the tcb:

word	bit	meaning
0		Claim word. Bit 1-11 of the tcbptr of the task that last sent a message to this task. Used to accept a sequence of messages from one task. 0000 means: ready to accept a message from any task (see sec. 1.2.1).
1		Head of receivequeue (see sec. 1.2).
2		Messagepointer of waited report. Used only when the task waits for a report. 0000 indicates that any report will be accepted (see sec. 1.2).
3		Head of reportqueue (see sec. 1.2).
4	0-3	Waitbits. When set they indicate a condition for which the task is waiting. 0 Wait for report. 1 Wait for message. 2 Wait for timeout. 3 Wait for code swap in (see sec. 1.3.3).
4	7-10	Task's priority (see sec. 1.3).
4	11	Stoppedbit (see sec. 1.3.2).
5		Runnable task: link in runqueue (see sec. 1.3). nonrunnable task: pointer to timeoutnode, if any (see sec. 1.9.6).
6	0	Ondiskbit. When set, taskcode is still on disk.
6	1	Set when task is reentrant.
6	5-11	Static tasknumber.
7	0-5	Virtual instructionfield. Number of the virtual memoryfield in which the taskcode is stored.
7	6-11	Virtual datafield (see sec. 1.4.2).
8		Ac.
9		Pc.
10		Mq.
11	0	Link.
11	1	Greater than flag.

11	3	Scheduling inhibitionflag. If set, the task runs with the scheduler inhibited (see sec. 1.6.3).
11	4	EAE-mode.
11	5	Usermode flipflop. Set only for protect tasks.
11	7-11	Stepcounter.
12		BASE.
13		X.
14	0-4	First page of taskcode.
14	6-11	Requested datafield (see sec. 1.4.2).
15		Blocknumber on disk.

Remarks.

Those bits for which no meaning was indicated are unused.

Word0 is used for the claimmechanism.

word0-3 are involved in the intertask communication.

word4-5 are used by the scheduler.

word7 (also bit6-11!) and word14 bit0-4 are undefined when the taskcode is not yet swapped in (i.e., as long as word6 bit0 is set).

word8-10 and word11 except bit2, bit3 and bit6 correspond to the hardware status. These registers are loaded into the hardware registers during the execution of the task. They are restored in the tcb when the task is interrupted or set to wait.

Word12 and word13 correspond to two locations in page 0 of the field in which the taskcode is stored. Their contents is loaded into these locations and restored in the tcb together with the loading and the restoring of the hardware registers. To the task it looks as if it has full access to these two locations in page 0.

As word0 of the entry in the stl points to the tcb as soon as a tcb is attached to the task, its previous contents must be saved in the tcb. Therefore word15 holds this contents.

Each time a tcb is attached to a task, it is prefilled with initial values. These values are:

word	bit	initial setting (octal)
------	-----	-------------------------

0		0000
---	--	------

1		0000
---	--	------

2		0000
---	--	------

3		0000
---	--	------

4	0	0
---	---	---

4	1	1
---	---	---

4	2	0
---	---	---

4	3	0
---	---	---

4	7-10	Priority
---	------	----------

4	11	0
---	----	---

5		Innocent pointer.
6	0	1
6	1	0
6	5-11	Static tasknumber
7	0-5	Undefined
7	6-11	Undefined
8		Undefined
9		Undefined
10		Undefined
11	0	Undefined
11	1	Undefined
11	3	0
11	4	Undefined
11	5	0
11	7-11	Undefined
12		Undefined
13		Undefined
14	0-4	Undefined
14	6-11	Undefined
15		Blocknumber on disk.

1.2 The intertask communication

The communication between tasks and their mutual synchronisation is based on the use of "messages". One task sends a message to an other task and after a while the receiver "reports the message" (back to the sender). A message is a blocklet of 5 consecutive locations, allocated in field 0. Messages are requested from and returned to an avallist system. About 150 messages may be requested. This is rather few compared to the 128 tasks that may be present in the system. Tasks must carefully use messages and return them as soon as possible.

The limited number of messages is one of the worst aspects of the MC8 system. We rejected two alternatives. One alternative is [2] to specify that tasks "call" other tasks. At the moment of the call parameters are passed from the caller to the callee. A disadvantage is, that the call "fails" when the callee is still active. Upon failure the caller will probably wait for a while and try again. This highly increases the danger of deadlocks. Moreover, it provides a rather poor scheme of communication. The other alternative is to store the messages in the code of the sendertask [3]. Tasks that want to send messages will never be confronted with a lack of messages, because it is their own responsibility to allocate them. Our difficulty is, that the sender and the receiver probably reside in different fields of the virtual

core. Multiple fieldswaps will occur when the receiver inspects and possibly changes the contents of the message. Field 0 is always present in the actual core, so that messages may be stored there without this objection.

Layout of a message:

word	bit	meaning
0	0	0: message need not be reported. 1: message need be reported.
0	1-11	Bit1-11 of tcbptr of the task that sent the message.
1		Link in receive- or reportqueue. 0000 indicates that the message has no successor.
2-4		User-defined information.

The size of a message (3 words of user-defined information) is sufficient for most common communication. For instance buffer transfers may be specified as:

word	bit	meaning
2	0	Read or write
2	1-5	Length in pages (max 4k)
2	6-11	Virtual field of the buffer
3		Starting address of the buffer
4		Other information.

Scheme of communication.

Assume task1 wants to set up communication with task2. Task1 requests a message, using the MSREQ pseudo instruction (see sec. 1.7.4). It stores the information in the message and executes a monitor command to send the message to task2. The monitor denotes the tcbptr of task1 in the message and either attaches the message to the receivequeue of task2, or directly activates task2. In either case task2 eventually "receives" the message and runs for a while. Meanwhile task1 may or may not continue to run, but sooner or later it would like to receive a report from task2. When task2 is done, it denotes its completion status in the message and executes a monitor command to report the message back to the sender: task1. The monitor attaches the message to the reportqueue of task1 or, if task1 was waiting for this report, directly activates task1. Finally task1 no longer uses the message. It returns it to the avallist system by executing the MSFREE pseudo instruction (see sec. 1.7.4).

Messages are identified in the system by specifying a pointer to their third word. This pointer is called the "message pointer" (msptr), and, as all messages are allocated in field 0, this pointer indeed identifies a message. Tasks also use this pointer to access the information in the message.

The monitor commands that have to do with message exchange are discussed below.

SNDMS

Send a message to a task.

Arg1: st# of receivertask.

Arg2: msptr of message to be sent.

Options: NONREP, DFPARM.

- A. Bit 1-11 of the tcbptr of the sender is denoted in the message.
- B. Bit 0 of word 0 of the message is set, unless the NONREP (nonreport) option was specified.
- C. If the receiver is waiting for the message, then
 - C1. the message is "received" and the receiver is runnable again else
 - C2. the message is attached to the receivequeue of the receiver. If the DFPARM option was specified, the actual datafield of the sendertask is copied into bit 6-8 of word 2 (the first informationword) of the message.

WTRP

Wait for a report.

Arg1: msptr of waited report.

Options: TIMEOUT, SWAPOUT.

First the reportqueue is checked to see if the message specified was already reported.

If so, the message is deleted from the reportqueue and its msptr is denoted in the task's ac (the task is not actually set to wait!).

If not, the msptr is copied into tcb[2] and the task is set to wait with tcb[4] bit 0 set.

If the argument = 0000, any message reported to the task will continue or reactivate it.

For the meaning of the options see below.

SNDWTR

Send a message and wait for its report.

Arg1: st# of receivertask.

Arg2: msptr of message to be sent.

Options: NONREP, DFPARM, TIMEOUT, SWAPOUT.

This is just an optimisation. The task might as well have executed the SNDMS and WTRP commands in this order, specifying the same msptr.

RP

report a message.

Argl: msptr of reported message.

- A. If bit0 of word0 of the message =0, then the message is returned to the avallist system (nonreport option).
- B. Else the tcbptr of the sender was denoted in the message.
- C. If the sender was waiting for this report, the msptr is stored in its ac and the sender is reactivated (runnable again). Else the message is attached to the reportqueue of the sender.

WTMS

Wait for a message.

Options: KEEP, TIMEOUT, SWAPOUT.

- A. If the KEEP option is not specified, the claim word of the task (tcb[0]) is cleared.
 - B. The receivequeue is inspected to see if an appropriate message is present.
A message is appropriate if the tcbptr of its sender (message[0] bit1-11) agrees with the claim word of the task. If nothing is indicated in the claim word (tcb[0]=0000), any message is appropriate.
 - C. If an appropriate message is present, then
 - C1. this message is deleted from the receivequeue and the task "receives" the message (the task is not actually set to wait!)
else
 - C2. The task is set to wait with tcb[4] bit1 set.
- For the meaning of the TIMEOUT and SWAPOUT options see below.

CHKRPO

Check reportqueue.

Argl: msptr of waited report.

This command is almost identical to WTRP. However, the task is not actually set to wait. If no appropriate report is found in the queue, the task is continued with ac=0. (Remember, that ac held the msptr if a report could be found.)

CHKRCQ

Check receivequeue.

Options: KEEP.

This command is almost identical to WTMS. However, the task is not actually set to wait. If no appropriate message can be found in the queue, the task is continued with ac=0. (Remember, that ac held the msptr if an appropriate message was found.)

Remarks.

"receiving" a message is accomplished in two steps:

First the tcbptr of the sender of the message is copied into tcb[0] of the receiveertask. The task is now claimed by the task that sent the message, and will remain so until it executes a WTMS command without specifying the KEEP option.

Next the msptr is stored in the task's ac.

The DFPARM option is a very ugly feature. It can be used to pass a fieldnumber in the message. As we pass the actual fieldnumber, this number might change before the receiveertask actually uses this number. This is prevented by locking the field whose number is passed (see sec. 1.4.4).

The option was added to facilitate communication with the system diskdriver (see sec. 1.9.2).

In word0 of a message and in tcb[0] bit1-11 of the tcbptr are used to identify the sendertask. This is indeed allowed, as by convention tcbptr's are always negative integers. Hence bit0 of a tcbptr is always set, and bit1-11 are sufficient to identify the tcb.

When a task sets itself to wait, it may always specify the TIMEOUT and/or the SWAPOUT option.

Specifying the TIMEOUT option (see sec. 1.9.6) guarantees that the task will be reactivated after some delay, even if the other waitcondition(s) (e.g., wait for report) are not fulfilled. The task may detect this and take appropriate action.

If a task expects that it will be waiting for quite a long time (e.g., several seconds) and if no important variables are stored in its code, it may specify the SWAPOUT option. If the task is actually set to wait, its code is erased from core. As soon as the task is runnable again, a fresh copy of its code is swapped in from disk. The tcb of the task is kept however, so that its registers are preserved. See also sec. 1.9.4.

1.2.1 Claiming tasks

In MC8 several processes may run simultaneously. A "proces" is an intuitive concept, which we are not going to define here. In this section we shall describe the way in which tasks are claimed in MC8.

When someone wants a job to be done under control of the MC8 system, a task is started that executes the job (termed maintask in this section). In simple cases this task just does what is needed without communicating with other tasks and finally executes the EXIT command.

A more complicated situation arises when the maintask wishes to delegate part of the work to other tasks (termed subtasks in this section). The maintask will send messages to subtasks in order to set

them to work. If the maintask needs only one message to specify to the subtask what it wants to be done, no problems arise. If however a sequence of messages is sent to a given subtask, messages from other tasks must be prevented from interspersing in this sequence. We say that the maintask must "claim" the subtask.

In fact a task is claimed each time it receives a message. Usually this claim terminates when the next WTMS command is executed, thus being invisible to the task. When the KEEP option is specified together with the WTMS command, the claim is not terminated and hence the subtask will only accept messages from the task that sent the former message.

Claiming is accomplished by writing bitl-ll of the tcbptr of the sender into tcb[0] of the receiver. When a claimable subtask is started, it starts processing messages, specifying the KEEP option at each WTMS command. When it recognises the last message of a sequence, it omits the KEEP option from the next WTMS command, thus enabling other tasks to send messages to it.

1.3 The scheduler

Tasks in the MC8 system either are runnable or are waiting for some or several conditions to be fulfilled. Each time the scheduler is activated, it selects one of the runnable tasks and executes the runchecks (see sec. 1.3.3). If one of this checks fails another task is selected. Else the status of the task is loaded into the hardware registers and the execution of its code is started up.

To each task a priority in the range $0 \leq \text{priority} \leq 10$ (octal) is attached. This priority is determined at assembly time. The only task of priority 10 is the idletask. System tasks and protect tasks have priorities in the range $1 \leq \text{priority} \leq 7$. Priority 0 is preserved for some monitor tasks.

The use of these priorities is opposite to the common interpretation, that is, tasks of lower priority are run first.

Runnable tasks are organised in priority runqueues. Each priority has its own runqueue. The runqueues are threaded through the task controlblocks (tcb[5]) of the runnable tasks.

When the scheduler is activated, it selects the first task of the priority 0 runqueue. If this queue is empty (no tasks of priority 0 are runnable), it selects the first task of the priority 1 runqueue. If that queue is empty too, the priority 2 runqueue is inspected etc. If all other runqueues are empty, the idletask, the first and only task in the priority 10 runqueue, is selected. The idletask is never set to wait and accesses only field 0, so that the runchecks cannot fail. This guarantees that the scheduler can always

select and start a runnable task.

Waitconditions expire as a consequence of a hardware interrupt (tasks waiting for some deviceflag) or as a consequence of the execution of a monitor command (tasks waiting for a message from an other task) *1. So during the execution of monitor commands or while treating hardware interrupts, tasks may be added to a runqueue. Therefore the scheduler is activated at the end of the interrupt section and after the execution of a monitor command. This insures that, if tasks having a lower priority than the task currently selected have become runnable, these tasks are selected first.

Some monitor commands cause the task that executes them (i.e., the task currently running) to be set to wait. The activation of the scheduler after such a command will select the next important task.

1.3.1 The routines OURUNQ and INRUNQ

Tasks are set to wait by calling the routine OURUNQ. This routine takes a pointer to the tcb of the task and one or more waitconditions or the stoppedcondition as parameter. It removes the task from its priority runqueue and sets up the appropriate bits in tcb[4].

As soon as a waitcondition for some task expires, an attempt is made to reinsert it in its priority runqueue. This is done by calling INRUNQ with a pointer to the tcb of the task as parameter. INRUNQ first clears tcb[4] bit0-3, erasing all waitconditions. But the stoppedcondition, if present, remains set. If the stoppedbit was not set, the task is now reinserted in its priority runqueue. Otherwise no further action is taken.

To optimise scheduling an integer, HPRIOR, and a boolean SCDREQ are used. HPRIOR generally holds the priority of the task currently selected. SCDREQ is usually false, indicating that no schedule request is pending. When INRUNQ inserts a task in its runqueue, it compares the priority of the task to HPRIOR. If this priority is higher or equal than HPRIOR, subsequent scheduling is obsolete as the current task will again be selected. If this priority is lower, subsequent scheduling is needed. SCDREQ is set to true and the new lowest priority is stored in HPRIOR.

Before entering the scheduler, the boolean SCDREQ is tested. If it is false, no scheduling is needed and the execution of the current task is continued. If SCDREQ is true, it is reset to false and the scheduler reinspects the runqueues starting at the one indicated by

*1 For exceptions see sec. 1.9.4 and 1.9.6.

HPRIO. Each time an empty queue is found HPRIO is incremented.

Note: If a monitor command sets the current task to wait, scheduling is required independant of the value of SCDREQ. In that case the scheduler is activated without testing SCDREQ.

1.3.2 The monitor commands STOP and RESUME

Generally speaking it is undesirable that tasks block and unblock each others execution. A proper synchronisation of tasks is based on message exchange. To cope with casualties the STOP command is introduced.

Task1 blocks the execution of task2 by executing the STOP command with proper argument. The following occurs:
If task2 was runnable (no waitbits nor the stoppedbit set in tcb[4]), OURUNQ is called specifying the stopcondition.
If task2 was already set to wait, the stoppedbit is turned on in tcb[4].

As long as the stoppedbit is set in tcb[4], task2 won't be reinserted in its runqueue (see above), though one of the waitconditions may expire. So the execution of task2 is effectively blocked.

To remove the stoppedbit from tcb[4] some task must execute the RESUME command specifying task2. The monitor removes the stoppedbit from tcb[4], erasing the stopcondition. If no waitbits are set in tcb[4], INRUNQ is called to reinsert task2 in its runqueue. If the stoppedbit was not set at all, the RESUME command has no effect.

Note: a side effect of stopping a task is, that it is removed out of the timeoutqueue (see sec. 1.9.6) if it was in. So any timeout is terminated when the task is resumed.

1.3.3 The runchecks.

Before the scheduler can start up the execution of the runnable task it has selected, two additional checks, the runchecks, must be made. These are discussed below.

1. Runcheck1: the taskcode must be in core.

If this check fails, the task is set to wait with tcb[4] bit3 (wait for code swap in) set. A message is sent to the taskfetcher specifying that this task must be swapped in.

As the task is now removed from the runqueue, subsequent scheduling will select some other task. Eventually the taskfetcher will swap in the code and make the task runnable again.

2. Runcheck2: the fields accessed by the task must be in the actual core.

If this check fails, the scheduler issues a swaprequest for the field, if possible (see sec. 1.4.6). Then it selects the next important task as if the previous one was not in the runqueue. Special care is taken for reentrant tasks (see sec. 2.2). Eventually the field will be swapped in the actual core and the scheduler will then select and start up the task that was skipped over (see sec. 1.4.6).

1.4 The virtual core

In the MC8 system a virtual core of upto 256k words (=64 fields of 4k words) is implemented in an actual core of 16-32k words (=4-8 fields). The number of virtual fields is set by the config parameter VIRMAX, $0 \leq \text{VIRMAX} \leq 63$. The number of actual fields is indicated in the config parameter ACTMAX.

To each virtual field an entry in the "fieldtable" (fldtab) corresponds. From the contents of the fldtab it can be determined whether a virtual field is in the actual core and, if so, in which actual field it is. To each actual field an entry in the coremap corresponds. Which virtual field is in a given actual field is denoted in the coremap. An actual field always contains some virtual field.

To each virtual field an area on the system disk, a "fieldslot", corresponds. When the virtual field is not in the actual core, its contents is stored there.

1.4.1 Paging algorithm

When a task wants to access a virtual field that is not in the actual core, a "swaprequest" may be done. That is, a request is done to swap the field under concern into one of the actual corefields. Before this can be accomplished the system must decide which of the actual fields must be evacuated. The algorithm that makes this decision is termed the paging algorithm.

Many paging algorithms have been published in literature [6]. As the PDP-8 has no special hardware for the implementation of a virtual core, most of these algorithms are rather time consuming or hard to implement on a PDP-8. We chose a very plain one.

The decision which virtual field is to be swapped out is made by the routine VFLESS. Each time it is called it either fails or succeeds. If it succeeds it delivers an integer indicating which actual field is to be evacuated. VFLESS uses a global integer,

VFNEXT, that wraps around in the range $0 \leq \text{VFNEXT} \leq \text{ACTMAX}$. VFNEXT is unchanged between two calls. The procedure LOCKED checks whether the virtual field, that is stored in the actual field indicated by VFNEXT, is "locked". If so, that virtual field must remain in core (see sec. 1.4.4). VFLESS fails if all fields are locked. This is the algorithm stated in ALGOL 68:

```
'INT' ACTMAX = #highest actual field#,
[0:ACTMAX] 'INT' COREMAP := #some initial setting#,
'INT' VFNEXT := 0,
'PROC' LOCKED = ('INT' I) 'BOOL':
    #virtual field stored in actual field I is locked#;
'PROC' VFLESS = ('REF' 'INT' ACTFLD) 'BOOL':
    ('INT' I := -1, 'BOOL' FOUND := 'FALSE';
    'WHILE' (I += 1) <= ACTMAX 'AND' 'NOT' FOUND
    'DO' VFNEXT := (VFNEXT + 1) 'MOD' (ACTMAX + 1);
        'IF' FOUND := 'NOT' LOCKED(VFNEXT)
        'THEN' ACTFLD := VFNEXT
    'FI'
    'OD';
FOUND)
```

This algorithm seems quite reasonable. It guarantees that the virtual fields most used are in the actual core most of the time. A disadvantage is, that if a task accesses more fields, say FA and FB, then FA is perhaps swapped out in order to swap in FB, which is useless. This objection can easily be removed at the cost of some core in field 0.

1.4.2 Datafield and instructionfield

Programs in the PDP-8 have access to two memoryfields: the datafield and the instructionfield. These fields are accessed via two 3-bit registers, the datafieldregister (dfr) and the instructionfieldregister (ifr). The instructions of a program are always fetched from the instructionfield. Special instructions are provided to change the contents of these registers. CDF changes the contents of the dfr and CIF changes the contents of the ifr.

Just like normal programs tasks have access to two fields: the virtual datafield and the virtual instructionfield. Before the execution of a task is started or resumed, the scheduler loads the dfr and ifr with the actual fields in which the corresponding virtual fields reside. If this is impossible because one of these fields is not in the actual core, runcheck2 fails and appropriate action is taken (see sec. 1.3.3). Changing the virtual fields that a task may access is slightly more complicated as it is for ordinary programs.

When designing MC8 we made the decision, that the code of a task should fit in (and is always stored in) one memoryfield (see sec. 1.1.1). This implies that the instructions of a task are fetched from the same virtual field as long as its code is stored in the virtual core. Changing the virtual, and hence the actual instructionfield is thus unnecessary and forbidden.

Changing the virtual datafield may be useful, for instance to access buffers in fields different from the virtual instructionfield. In fact, experience shows that large programs and programs requiring large buffers fairly often use the CDF instruction. Before a task can change the contents of the dfr, it has to find out in which actual field the desired virtual field resides, if it even is in the actual core. This means that the task has to inspect system tables, which is unadvisable. Moreover, the task might be interrupted halfway its investigations and the system tables might be changed by the time it is resumed.

The simplest alternative is to introduce a monitor command to change the datafield of a task and let the monitor do what is needed. Activating the full mechanism of a monitor command each time a task wants to change its datafield, would however be too inefficient. To obtain an easy to use and efficient implementation we introduced 3 pseudo instructions and 1 related monitor command. The table below gives the time estimates for each pseudo instruction compared to the time that the execution of the actual CDF instruction takes.

pseudo ins time estimate	
CDFCUR	5
VRCDF	6
VCDF	28

The effect of the pseudo instructions.

CDFCUR Sets the datafield equal to the instructionfield.

VCDF Takes the contents of the next location as an argument, holding the number of the desired virtual field. If this field is in the actual core, the dfr is set to the corresponding actual field. Otherwise runcheck2 fails. An emergency escape (see sec. 1.7.7) is then used and similar action as described in sec. 1.3.3 is taken.

VRCDF Sets the datafield to the "requested" (see below) virtual field. Each first time this pseudo instruction is encountered after start or restart of the task's execution, a check similar to that described for VCDF is made. If it succeeds one extra side effect occurs: the routine implementing this pseudo instruction is patched with a "hard" CDF instruction to the correct actual field. Subsequent calls on this routine will cause the correct CDF instruction to be executed immediately without any check. The patch is removed when the task is

interrupted or executes a monitor command. This makes the VRCDF pseudo instruction extremely powerful in cases where a burst of CDF's to the same virtual field occurs.

A virtual field is "requested" by executing the monitor command REQCDF. The desired virtual field is indicated in the rightmost 6 bits of the ac. It is denoted in tcb[14] bit6-11.

None of the above pseudo instructions touches the ac, link etc. (neither does the actual CDF instruction).

A special remark need be made for changing the datafield to virtual field 0. This can be achieved by executing a "hard" CDF-0 instruction, as virtual field 0 is always present in actual field 0. Tasks wishing to access information stored in messages may benefit from this.

Another related pseudo instruction need be mentioned here. The VRDF pseudo instruction clears the ac and stores the number of the current virtual datafield in bit6-11 of the ac. It is more or less equivalent to the hardware RDF instruction.

1.4.3 Fieldtable and coremap

Information on each virtual field is stored in the fieldtable (fldtab). Each field has a 1-word entry in this table:

bit	meaning
0	1: Field in actual core. 0: Field on disk.
1	Set if the field is in the actual core.
2	1: Whole field used for datastorage. 0: Otherwise.
3	1: One-task-only part of this field is used (see sec. 1.7.2). 0: Otherwise.
4	Set if field in the actual core.
5	Coreresidentbit (see sec. 1.4.5).
6-8	Indicate the actual field in which the field is stored. Undefined if field on disk.
9	Moninbit. Set if and only if a copy of the general page 0 is present in the field (see sec. 1.7.8).
10	Datapresentbit. Used to optimise swapping (see sec. 1.9.2).
11	1: Field is in use (code or data may be stored in it). 0: Field is free.

To some extent the coremap can be viewed as a cross reference table for fldtab. Each actual field has a 1-word entry in the coremap:

bit	meaning
1-5	Lockcount. Counts to 31 locks (see sec. 1.4.4).
6-11	Virtual field that currently resides in this actual field.

1.4.4 Locking and unlocking fields

Sometimes it is desirable (or even necessary) to insure that a virtual field remains present in the actual core for a shorter or longer period. Asserting this is termed "locking" a field. The opposite is termed "unlocking". To prevent fields from being swapped out of the actual core the procedure LOCKED in the paging algorithm (see sec. 1.4.1) checks the lockcount (see sec. 1.4.3) of the field that is about to be swapped out. If the lockcount is nonzero, this field will not be swapped out and an other field will be tried.

We list a few "lockconditions" (reasons to lock a field in core):

- A. Tasks are allocated in the field, that are activated so often that it is preferable to have them permanently in core.
- B. Field 0, containing the coreresident portion of the system, must remain in core.
- C. I/O processes may access a field so often that it is undesirable (or, in case of real time restrictions, impossible) to swap it out.
- D. Swapping out a field while a databreakproces accesses it, causes serious errors.
- E. Fields containing routines connected to an interruptslot must remain in core (see sec. 1.5.4).

Lockconditions may overlap one another. Therefore the lockconditions that apply to a given field are counted in the lockcount. When a lockcondition is initiated, the field is locked, which is equivalent to incrementing its lockcount. When a lockcondition terminates, the field is unlocked, which is equivalent to decrementing its lockcount. The field can be swapped out again, as soon as its lockcount is reset to 0.

Locking for conditions A and B is automatically performed by the monitor. Locking for other conditions must be treated by the tasks that give rise to these conditions.

The monitor command LOCK locks the datafield of the task that executes the command in core. Note: that this field is indeed in the actual core, as otherwise runcheck2 would have failed, and hence the task would not have been executed.

The monitor command UNLOCK unlocks the datafield of the task that executes the command.

1.4.5 GETFLD and FRFLD

Initially field 0 is the only field in use by the system. All other fields are free. Field 0 holds the coreresident portion of the system and is locked in actual field 0. As the space occupied by the system will never be considered as free, field 0 will never be "returned" and hence remain in the actual core as long as MC8 is running.

Sooner or later tasks and buffers must be allocated in the virtual core. If this cannot be done in one of the fields that is already in use, GETFLD is called to select one of the free fields. The selected field is no longer free; bit11 of its entry in fldtab is set. When calling GETFLD the coreresident condition may be specified. If so, the coreresidentbit (bit5) will be set in the entry in fldtab. As soon as the field is actually swapped in, the coreresidentbit is checked and if it is set the field is locked.

When a buffer is returned or when a task executes the EXIT command or specifies the SWAPOUT option, the space occupied by such a buffer or by the taskcode is considered as free again. If in this way a whole virtual field is free again, it is "returned". That is, FRFLD is called with the corresponding virtual field number as parameter. If the coreresidentbit of the field was set, the field is first unlocked. Next bits 2, 3, 5, 10 and 11 of the entry in fldtab are cleared, marking the field as free.

1.4.6 Swapping

The generation and treatment of swaprequests (swreqs) is closely interrelated with the operation of the scheduler. On one hand the scheduler, on failure of runcheck2, can issue a swreq. On the other hand the way in which swreqs are treated heavily depends on the priority mechanism embedded in the scheduler. The order in which fields of the virtual core are swapped in the actual core severely influences the performance of the system.

Each task can access at most two fields at a time: the instructionfield and the datafield. Both these fields must be in the actual core when the task is executed. Here follows the easy way to achieve this.

Simple field swap scheme.

Assume some task, say T1, wants to access a field that is not in the actual core. The scheduler will detect this and issue a field swap. Next it waits until the field swap is completed (about 0.15 seconds) and then starts up the execution of T1.

This scheme has the disadvantage that the system wastes time during field swaps. Probably other tasks could have been executed that access fields that do reside in the actual core.

More concise scheme of field swaps.

The next scheme removes the disadvantage mentioned above, but it introduces great danger for deadlocks.

When the scheduler selects a task whose fields are not in the actual core, it appends a swreq to a queue. Next it continues inspecting runqueues, adding swreqs to the queue as required. Finally it hits on a task that can be executed. This certainly happens as the idletask accesses only field 0, which is locked in core.

Parallel to this process of task selection and task execution another process, the swapper, eats nodes from the swreq queue and performs the indicated field swaps. Each time the swapper completes a field swap, it interrupts the task currently in execution and sets the schedulerrequestflag. The scheduler now reinspects all runqueues and probably starts up the task that caused the field swap just completed.

The example below demonstrates the danger for deadlocks.

Assume there are 4 virtual fields: VF0, VF1, VF2 and VF3. The actual core has 3 fields initially containing VF0 (locked), VF2 and VF3. Apart from the idletask there are two runnable tasks T1, accessing VF1 and VF3, and T2, accessing VF2 and VF3. In the sequel we indicate a swreq by first denoting the field to be swapped out and next the field to be swapped in. When the scheduler is activated, the following might occur.

- A. Select T1.
- B. Swreq VF2-VF1 to queue.
- C. Select T2.
- D. Swreq VF1-VF2 to queue.
- E. Select idletask.

Note: Of course tasks are not allowed to access fields that are halfway being swapped in or out. As soon as the first field swap is completed, T1 will be reselected. Although its fields are now in core, it cannot be started up, as the next field swap (which is about to be performed by the swapper) will swap out VF1 again. Hence step A is retaken and so forth.

This is the very reason why organising swreqs in a queue is a bad principle. Things go better when we restrict the length of the queue to 1. This is effectively done in the MC8 system.

Field swap scheme of MC8.

- A. The scheduler selects a runnable task.
- B. If both its fields are in the actual core, the task is executed, otherwise step C is taken.
- C. If a swreq is still pending then goto step E.
- D. Issue a swreq, eventually starting the swapper.
- E. Schedule the next important task and goto B.

When the field swap is completed, all runqueues are reinspected, effectively retaking step A. Probably the task that caused the swreq, say T1, will now be executed. It can however occur, that a task, say T2, of a lower priority than T1 has become runnable. T2 is more important than T1 and thus will be selected first. If its fields are not in the actual core, the scheduler might issue a swreq such, that the fields of T1 are swapped out again. This tedious event can happen repeatedly, but ultimately the task that caused the swreq must have priority 0. This task will certainly be executed at the completion of the swreq.

Summarising it turns out, that in this scheme field swaps may be done in a very unlucky order, but tasks of a lower priority will always get their jobs done. This is reasonable in a priority scheme.

Detailed description of the MC8 field swap scheme.

At the selection of a task or at the execution of one of the change-datafield pseudo instructions (step A) the routine FCHEX is called to check whether the desired field is in the actual core (step B). If it is, the execution of the task is started up or continued respectively. Otherwise if no swreq is pending (step C) a swreq is issued (step D). This is achieved by sending a special message, the swapmessage, to the system diskdriver. In this message the field to be swapped out (determined by calling VFLESS) and the field to be swapped in (the desired field) are specified. Moreover, the "ondisk"condition of the field to be swapped out is set, to prevent subsequent access from other tasks, and the swreqpendflag is set.

Now the scheduler can continue operating. Sooner or later (perhaps immediately) the system diskdriver will be scheduled and accept the swapmessage. This certainly occurs as the system diskdriver has priority 0 and never waits for other tasks. It just accepts a message, executes the indicated transfer and waits for the next message.

When the system diskdriver has accepted the swapmessage, it exchanges the indicated virtual fields and reports the swapmessage. As word0 of the swapmessage always holds the tcbptr of the swaptask, the message is now reported to this task.

The swaptask is a nonterminating loop. It waits for the report on the swapmessage, clears the "ondisk"condition of the field just

swapped in, updates its general page 0 (see sec. 1.7.8), clears the swreqpendflag and sets itself to wait again. Note: no swreq can be done until the swaptask has set itself to wait.

As the swaptask has priority 0, all runqueues will be reinspected when it executes the WTRP command.

Remark.

If VFLESS fails no swreq is issued. This means, that, if all fields are locked, no swreqs are done. The idletask will probably be started up. Sooner or later a proces locking a field will terminate. At that moment the runqueues are automatically reinspected from that priority downward. If this causes a swreq, everything is o.k., however if it does not the system may remain in high priorities unless some external event makes a priority 0 task runnable. This will probably be the system timer started by the clock.

Multiple fieldswaps will occur if all fields but 1 are locked and neither the instructionfield nor the datafield of a selected task is in the actual core. This is the bad situation in which the system keeps exchanging data- and instructionfield until one of the other fields is unlocked. Therefore it is advisable to have at least 4 actual fields. Field 0 is always locked. Coreresident tasks can be stored in field 1, which will then remain locked for quite a long time. The remaining two actual fields are available for the normal virtual core mechanism.

1.5 The interrupt section

The interrupt mechanism of PDP-8 computers is rather poor. Certain external events (a flag raised by a peripheral device interface) or internal events (executing I/O instructions with the usermode flipflop set) can cause an interruptrequest (intreq). To each event a flag is connected. These flags are termed deviceflags and userflag respectively. (In the sequel we will mention only deviceflags, where both deviceflags and userflag may be meant.) Causing an intreq is always accompanied by setting a deviceflag.

Some devices can be "disabled". When a flag of such a device is raised, it does not cause an intreq. Devices that are not disabled are termed "enabled".

To summarise:

An intreq is pending if and only if at least one deviceflag of an enabled device is raised.

When a program runs with the interruptsystem turned on (ION), an intreq is granted by an interrupt. The following occurs:

1. The current values of ifr, dfr and the usermode flipflop are stored in the so-called saveregister.
2. The value of the pc is stored in location 0000 of field 0, INTPC.
3. The interruptsystem is turned off (IOF, deaf).
4. A special part of the program, the interrupt section, is entered at location 0001 in field 0.

The interrupt section has to find out which device caused the interrupt. This is done by interrogating the deviceflags one by one. Once a raised flag of an enabled device has been found, it is cleared or the device is disabled to remove the intreq and the device is serviced. Now the interrupted program can be resumed.

1.5.1 The MC8 interrupt section

1. Upon entrance of the interrupt section the status of the interrupted program is temporarily saved.
2. The skipchain is rushed through to find a raised deviceflag of an enabled device. Such a flag must be present.
3. When a flag is found, some device-service routine is called. This routine either clears the flag or disables the device. Moreover, it may cause side effects to communicate with tasks in the system. All device-service routines end with calling the routine RPINTR. The location after the call either holds 0000, specifying that no message need be reported, or the msptr of the message to be reported.
4. After this call, the intreq will probably be removed. This is uncertain however, as meanwhile flags of other enabled devices may have been raised. Some PDP-8 computers allow to interrogate the "interrupt request line" to see if another intreq is still pending. If so, step 2 is repeated.
5. By now the interruptsystem can be turned on again. A decision must be made either to resume the interrupted program or to activate the scheduler in order to start up some task (see sec. 1.6.3).

1.5.2 The skipchain

The skipchain is a piece of code, that is rushed through in one direction (no jumps back). It is used to find a raised flag of an enabled device. The skipchain is built of so-called interruptslots (intslots): sequences of 5 locations corresponding to one deviceflag. Each deviceflag has its own intslot.

The first instruction of an intslot is a skipiot, skipping the next instruction if the corresponding flag is set. If it is not, the

program must interrogate the next deviceflag, hence the second instruction is a "jmp .+4" to jump to the first instruction of the next intslot. The remaining 3 locations of an intslot check if the device is enabled and, if so, either constitute or activate the device-service routine.

The order in which intslots are arranged in the skipchain (i.e., the order in which deviceflags are tested) is very important, as this represents a software priority arbitration. Moreover, a bad arrangement may cause considerable (or even intolerable) overhead. E.g., the intslot of a device causing many interrupts must not be located at the end of the skipchain. The same goes for a device (e.g. DECTape or RK disk) requiring fast servicing. The order of the intslots is determined at assembly time by the contents of the config file (see appendix A).

1.5.3 Intslots and related monitor commands

The contents of the first two locations of an intslot is steady. They hold the skipiot and a jump to the next intslot. The skipiot is devicedependant and may be used to identify the intslot. The contents of the remaining locations depends on the status of the corresponding device (enabled/disabled) and of the way in which communication with a task is established.

The 4 possible states of an intslot and the monitor commands that set intslots in these states are discussed below. All these commands are executed with the skipiot in ac to identify the intslot to be changed. The other devicedependant parameter that must sometimes be specified (FRINTR and CLINTR) is the cleariot. The cleariot is an instruction to clear the deviceflag. If specified, it is passed in arg1.

As intslots establish the communication between tasks and peripheral devices, their contents must exclusively be controlled by one task at a time. The access of different tasks to an intslot must somehow be synchronised. In this first version no attempt is made to coordinate such access. Tasks have to be decent! In future versions another monitor task, the claimtask, is planned. Tasks then need the explicit permission of the claimtask, before they are allowed to access an intslot.

Free intslots.

When no task in the system is interested in the interrupts caused by some device, and yet the device is enabled, the corresponding intslot must be set to free. The only thing that need be done when such a device interrupts, is clearing its deviceflag.

The contents of the last three locations of the intslot is:

CLEARIOT /clear deviceflag

JMS RPINTR /clears ac and no report.

0 /when called with zero argument.

The monitor command FRINTR sets an intslot to free.
arg1 =cleariot.

Disabled intslots.

If a device is disabled, the program must continue rushing through the skipchain, even if it finds the corresponding deviceflag set. This device did not cause the interrupt. Calling RPINTR when the program reaches the intslot of a disabled device would prevent proper device servicing and hang up the system.

The third instruction of a disabled intslot is "jmp .+3", that is a jump to the next intslot.

The DISINTR command sets the intslot to disabled. It uses no arguments.

Claimed intslots.

The choice of the name "claimed" is somewhat misleading. When an intslot is claimed, a message is attached to it, that is reported each time the corresponding device interrupts. The task that attached the message to the intslot may recognise the interrupts by inspecting its reportqueue. It may wait for the interrupt by executing the WTRP command (see sec. 1.2). This is the common way of synchronisation between a task and a peripheral.

The contents of the three last locations of a claimed intslot is:

CLEARIOT /clear flag, possibly read status

JMS RPINTR /store status in message and report it

mmmm /msptr of message to be reported

The routine RPINTR fetches the msptr, reports the message and jumps to FSTEXT. If no msptr is present (free intslot), control is immediately sent to FSTEXT. FSTEXT is the spot where we are about to leave the interrupt section (sec. 1.5.1 step 4).

The command CLINTR attaches a message to the intslot.

Arg1 =cleariot,

Arg2 =msptr.

The communication and synchronisation using a claimed intslot might go as follows:

- A. Request a message (MSREQ pseudo instruction).
 - B. Execute the CLINTR command, specifying the desired device, the desired cleariot and the msptr.
 - C. Activate the device, using device dependant code.
- The device is now busy and will interrupt when it is ready to receive new directives.

- D. The task may continue to run, executing the CHKRPQ command from time to time (polling) to see whether the device has already interrupted, or it may execute the WTRP command to wait until the interrupt occurs.
- E. When the device interrupts, the cleariot is executed, possibly reading a status in ac, ac is stored in the message and the message is reported.

Example. Communication with a keyboard.

```

KCC          /clear keyboardflag
IAC;KIE;CLA  /enable device
MSREQ        /request message
DCA MSP      /store in arg2
TAD MSP      /fetch msptr again
DCA MSP1     /store for future use
TAD (KSF     /skipiot in ac
CALL;CLINTR  /execute CLINTR command
      KRB    /arg1 =cleariot
MSP, 0       /msptr is stored here
...
KRD, 0       /entry of read char routine
CALL;WTRP    /wait for report
MSP1, 0      /arg1 =msptr
Cla         /remove msptr from ac
CDF 0
TAD I MSP1   /fetch char from message
CDFCUR
JMP I KRD

```

Connected intslots.

A more concise communication between a task and a peripheral can be established by connecting a routine to the corresponding intslot. In principle this routine is called each time the corresponding device interrupts. The routine must be stored in a locked field.

The contents of the last three locations of the intslot is:

```

CIF CDF FLD  /ifr and dfr to field of routine
JMS I .+1    /subroutine call
ssss        /pointer to subroutine entry

```

The CNINTR command connects an intslot and a routine.

Arg1 =pointer to routine entrypoint.

Arg2 =msptr.

Dfr =field in which the routine resides.

The message whose ptr was passed in arg2 may be used by the connected routine to report to the task. If no such message is used, arg2 must be =0.

From the instructions listed above it is seen, that the routine is called not only when the device interrupts, but each time the

intslot is checked and its skipiot skips. Therefore the routine is called as a proper subroutine. If the device is not enabled, and thus the routine is called without reason, it can return via its subroutine link in order to continue inspecting intslots. Note that this link must be incremented by 1 before using it and that ifr and dfr must be reset to 0 by a "CIF CDF 0" instruction. If the device is enabled, it is the responsibility of the routine to clear the flag or to disable the device. The routine must end by calling RPINTR. At the moment of the call ifr must be set to 0 (by a "CIF 0" instruction) and dfr must be set to the current field (in order to insure a proper argument handling by RPINTR). A special macro was introduced for this purpose: IEXIT. It calls RPINTR via a pointer on the general page 0. The end of a connected routine could look like:

```
CIF 0
IEXIT
msptr  /or 0000.
```

1.5.4 Connected routines

As stated above it is possible to connect a routine to an intslot. This routine is called to service a device each time it interrupts. Several restrictions apply to such routines.

- A. As devices interrupt at unpredictable times and as the routine is called without any further check, the task that connects the routine must ensure that the routine remains in the actual core. This is done by locking the field in which the routine resides.
- B. If the corresponding device is disabled, the routine must return control to the skipchain, as otherwise intslots closer to the end of the skipchain will be serviced incorrectly.
- C. As the routine is directly called from the skipchain, it is not allowed to turn on the interruptsystem. Nor is it allowed to use any variable or call any subroutine that is at the same time accessible from any task. Especially the general page zero must be used with care.
- D. It is undesirable to have the interruptsystem turned off (to be deaf) over a long period. As the machine is deaf while executing connected routines, these routines must be as fast as possible.
- E. As connected routines do not use system calls to return, they must carefully set the registers ifr and dfr before returning. Upon entrance ifr and dfr refer to the field of the routine. At return ifr must be set to 0, and dfr must refer to the field of the routine, as it is used to fetch the msptr.

Connected routines are used to speed up task-peripheral communication. We give two examples.

The RK disk.

The RK disk control raises its flag as soon as the transfer of one block is completed. If the next block is to be transferred too, this transfer must be initiated within 100 microsec. After that time the blockheader has passed under the heads, so the transfer of the next block is delayed by one revolution of the disk. As scheduling takes about 800 microsec, it would take too long to schedule a drivertask upon recognition of the disk interrupt. The job must be done by a connected routine. The drivertask starts up the transfer and specifies to the routine the number of blocks. When a disk interrupt occurs, the routine initiates the transfer of the next block. When all blocks are transferred, it calls RPINTR to report a message to the drivertask.

Buffered I/O.

Some devices (e.g. a fast papertape reader) cause so many interrupts, that scheduling at each interrupt would give too much overhead. A driver for such a device might hand a buffer to a connected routine. The routine would handle the I/O and report a message as soon as the buffer is filled.

Generally a connected routine can be viewed as a loop. Each time the corresponding device interrupts one cycle through the loop is made, terminating with the sequence:

CIF 0; IEXIT; 0

when the loop has been executed a number of times, some endcondition will be detected. This may be visualised to the outside by reporting some message:

CIF 0; IEXIT; msptr

To get the loop started, it is often easy to enter it halfway as if an interrupt had occurred. A task cannot simulate the occurrence of an interrupt. If a task jumps into a connected routine, serious errors occur at the end of the interrupt section. Therefore we introduced the monitor command SIMINTR.

The SIMINTR command requires a pointer in the loop as an argument and assumes that dfr refers to the field of the connected routine. The task seems to be interrupted at the point where it executed the SIMINTR command. The connected routine is entered at the spot indicated by the argument. When the routine executes the IEXIT macro the normal procedure at the end of the interrupt section will be performed. The task is resumed as soon as it is scheduled.

1.5.5 Changing the state of an intslot

After this description of the commands that control the contents of the intslots, we will point out the possibilities of this mechanism. It will turn out that changing the enabled/disabled status of a device is a delicate action. Sometimes it can only be accomplished within a connected routine. When trying to implement a more concise control mechanism, one hits on the problem of the very baroque design of many device interfaces for the PDP-8 computers. Especially the enable and disable instructions often have undesired side effects on the performance of a device, and require strange bitsettings in the ac.

Each device in the system has a default status (enabled or disabled). This status is determined at assembly time *1.

When a given device is not used by any task, the corresponding intslot must be in the free state if the device is enabled, and in the disabled state if the device is disabled. Initially all devices, except those used by monitor tasks, have the default status and their intslots are free or disabled respectively. It is good practice that a task returns a device (and its intslot) in the default status after using it.

To use a device in its default status is straight forward. The commands to change the intslot can be executed without harm. Using a device in the nondefault status implies that its status must be changed. This is complicated, as at each instant of time the state of the intslot must be in accordance with the status of the device. The intslot of an enabled device must be free, claimed or connected. The intslot of a disabled device must be connected or disabled. As the enable or disable instruction and the change intslot command cannot be executed simultaneously, it is hard to fulfill this requirement.

First we note, that a possible disagreement between the status of a device and the state of its intslot can only be detected (and then blows up the system) when the corresponding skipiot skips. Consequently no errors occur as long as the deviceflag is cleared. So, changing both the status of a device and the state of its intslot is allowed within a part of the program where the deviceflag remains cleared.

*1 Usually the default status of a device is chosen equal to the status reached on receival of the "initialisation pulse". The "initialisation pulse" is sent by depressing the "start"- or "clear"-button on the console of the machine, or by executing the CAF instruction.

If no such part exists, the problem may be solved as follows. First a routine is connected to the intslot, using the CNINTR command. This routine uses a software flipflop to indicate the status of the device. If this flipflop is set to disabled, the routine returns immediately to the skipchain, else it services the device and jumps to FSTEXT. Initially the setting of the flipflop corresponds to the default status of the device. Now the above requirement reduces to the requirement that the setting of the flipflop agrees with the status of the device. But, as a possible disagreement can only be detected by calling the routine, it suffices to guarantee that the routine is not called during a period of disagreement. This can be achieved by changing both the flipflop and the status of the device either within one subroutine call, or in a section accessed using the SIMINTR command.

Note: in some cases a software flipflop is unnecessary. Some devices (e.g. DECTape) allow programs to interrogate their status. In such cases a connected routine can always decide between returning to the skipchain and servicing the device. Changing the status of the device then cannot cause troubles.

1.6 Flow of control

1.6.1 Execution states

To clarify the discussion below we distinguish 5 execution states. At each instant of time the system is in exactly one of these states. Sometimes the decision when precisely the system transides from one state to the other is somewhat arbitrary.

The execution states are:

- A. Executing the code of some task.
- B. Scheduling. Selecting or starting up some task.
- C. Executing some pseudo instruction (i.e., executing the code of some routine, that implements a pseudo instruction).
- D. Executing a monitor command.
- E. Executing code of the interrupt section.

Note: monitor tasks, like other tasks, are executed in state A.

1.6.2 Task switching

In the states A, C and D the system is dedicated to one task: the current task. It is either directly executing the current task's code (state A), or it is performing a pseudo instruction (state C) or monitor command (state D) for the current task.

The system will remain dedicated to the current task, until:

1. At the end of a monitor command the current task has set itself to wait; some other task must be selected.
2. At the end of a monitor command a task, more important than the current task, appears to be runnable; the more important task must be selected.
3. At the end of a pseudo instruction runcheck2 fails (the new datafield is not in the actual core); some other task must be selected.
4. After treating a device interrupt a task, more important than the current task, has become runnable; the more important task must be selected.

At such a moment the system has to switch tasks, save the current task's status in its tcb, and activate the scheduler.

The first three situations are detected at well-defined spots in the monitor program. It is then easy to pick up the status of the current task, save it and enter the scheduler.

The fourth situation is detected at the end of the interrupt section. The proper action to be taken then depends upon the execution state of the system at the moment of the interrupt.

1. State A at the moment of the interrupt.
At the moment of the interrupt the system was executing the code of the task. Hence the status of the task was present in the hardware registers and saved at the start of the interrupt section. This status may be copied into the tcb of the task, whereupon the scheduler can be activated.
2. State B at the moment of the interrupt.
The system was scheduling when the interrupt occurred. Consequently the system was not dedicated to a special task. The scheduler can be activated without saving a status.
3. State C or D at the moment of the interrupt.
This is complicated. The system was halfway the execution of a monitor command or pseudo instruction when the interrupt occurred. Scheduling an other task, that subsequently might execute the same monitor command or pseudo instruction, inevitably leads to errors. Variables and subroutine entries will get overwritten and perhaps the contents of system tables will be lost.
Therefore during state C or D either the ION/IOF flipflop is set to IOF (postponing interrupts hardwarewise), or SCDINH is set (preventing activation of the scheduler at the end of the interrupt section).

Remark.

Some tasks use tcb[11] bit3 (see sec. 2.2) to run with SCDINH set. This inhibits scheduling caused by device interrupts and effectively overrides the normal priority scheme. In such cases task

switching only occurs at the end of some monitor call or at the end of some pseudo instruction.

1.6.3 Critical sections, scheduling inhibitionflag

Some sections of the system can be executed in more than one execution state. For instance, the section that saves the status of the current task may be executed at the end of a monitor command (state D), after the execution of some pseudo instruction (state C) and at the end of the interrupt section (state E). The message report routine may be called from the interrupt section (state E) and may be called to implement a REPORT command (state D).

Most changes in the execution state occur under control of the monitor program (albeit indirectly via a pseudo instruction), and do not cause troubles. Device interrupts (changes to state E) occur at unpredictable times. Critical sections (when executed in a state other than state E) must be protected against these interrupts. This protection is effected using the hardware (ION/IOF) flipflop and a software flag, SCHINH (scheduling inhibited).

To understand the use of these two flipflops, we divide the interrupt section into two parts: the first part contains the skipchain and the device-service routines (1.5.1 steps 1-4), the second part decides either to resume the interrupted program or to save the status of the current task and to activate the scheduler (1.5.1 step 5).

The simplest protection against device interrupts is turning the interruptsystem off (IOF). Intreqs are not granted by an interrupt; interrupts are postponed until an ION-instruction is executed to turn the interruptsystem on again. In this way one can exclude interrupts in critical sections.

Postponing interrupts for a rather long period is undesirable, as many devices like or need real time servicing. Therefore the second part of the interrupt section first checks the SCDINH flipflop. If it is set the interrupted program is always resumed. These sections that only conflict with part 2 of the interrupt section (status saving) are sufficiently protected by setting SCDINH and can be executed with ION. Devices can normally be serviced, but subsequent scheduling, if required, is delayed until SCDINH is cleared. (Note: this forces a test for SCDREQ at all places where SCDINH is cleared.)

In the listing in appendix A we tried to mark the critical sections. The interrupt section is marked with a "!" on each line. Sections of state C and D that are executed with IOF, because they conflict with the first part of the interrupt section, are marked with a "\" on each line. Some sections where the execution state

changes are marked with "-" on each line.

Summary.

Tasks (state A) are executed with SCDINH cleared and ION (but see remark in sec. 1.6.2).

The scheduler (state B) runs with ION. Loading the status of a task into the hardware registers conflicts with the status save routine and hence is done with SCDINH set. Just before task start-up, SCDINH is cleared and SCDREQ is tested.

Most pseudo instructions (state C) are executed with IOF. Using SCDINH would take too much time. If a datafield is not in the actual core, SCDINH is set, the interrupt system is enabled and the status save routine is called. This is achieved by executing an emergency call (see sec. 1.7.7).

Monitor commands (state D) are executed with SCDINH set. Sections that conflict with part 1 of the interrupt section are executed with IOF.

The interrupt section (state E) is of course executed with IOF. Part 2 can only be executed if SCDINH is cleared.

Remark.

Parts of the monitor, that are accessed by monitor tasks as well as by routines to implement monitor commands, constitute another area of code accessed in several execution states. When a monitor task is running and no other tasks can force scheduling, no problems arise. Either the monitor task is in execution, or the system performs a monitor command for it. These two cannot get mixed. As monitor tasks (except the idletask, which does not give conflicts) either have priority 0 or run with SCDINH set, no other tasks can interrupt them. Hence scheduling will not occur until a monitor task explicitly sets itself to wait!

1.7 The general page 0

The lowest page of each field, page 0, plays a special role in the PDP-8. From any other page in the field it may be accessed with a direct reference, whereas all other interpage communication requires indirect referencing. Therefore in many programs page 0 holds global constants, variables and subroutines. Moreover, locations 0010-0017 octal are in page 0. These locations are the so-called autoincrement registers. They are very useful in accessing data in linear arrays.

In the MC8 system tasks are relocated, so that they cannot be sure to be located on page 0. Hence they cannot benefit from the special properties of this page, and even worse, in some cases unmeant and unlucky use of the autoincrement registers might cause

serious errors. We decided that tasks are never located on page 0 (see sec. 1.1.1).

In order to preserve the special advantages of page 0 for the MC8 system, we used it for the following purposes:

- storage of task registers,
- storage of the one-task-only part,
- storage of general constants,
- the implementation of pseudo instructions,
- the entrance of the monitor command section.

1.7.1 Storage of task registers

In order to permit the use of two more global registers in a task, two locations on page 0 are used, namely BASE and X. They may be accessed from each page of a task. Their contents is saved and restored together with the contents of the hardware registers such as ac, pc etc. Besides that, BASE and X play a crucial role in group 2 pseudo instructions (see sec. 1.7.4).

In some PDP-8 machines a hardware mq register is not present. The usage of a number of features (e.g., the JUMS pseudo instruction) is much simpler when the mq is present. Therefore in those machines that lack a hardware mq we introduced a register similar to BASE and X, named MQ. The presence of a hardware mq can be determined from the type specification in the config file (PDP-8/I or PDP-8/E, EAE not defined or EAE=1).

To facilitate the use of this MQ register, we introduced the instructions STORMQ to load it, and GETMQ to fetch its contents. When a hardware mq is present, STORMQ is implemented as "MQL", GETMQ as "MQA". When a hardware mq is missing, STORMQ is implemented as "DCA MQ", GETMQ as "TAD MQ".

The effect of STORMQ is equivalent in both cases, apart from subsequent EAE instructions. GETMQ either 'or's or adds the contents of mq and ac, and is equivalent only if ac is cleared before executing this instruction.

1.7.2 The one-task-only part

If a task wants to use substantially more of page 0 than the two or three registers mentioned above, it may request for the one-task-only part of page 0. This must be specified at assembly time and results in setting the ZREQbit (zero request bit) in the task's entry in the stl. While swapping this task into the virtual core the task fetcher recognises this bit, whereupon it allocates the task in a field in which the one-task-only part of page 0 is still free.

Whether the one-task-only part of page 0 of a given virtual field is free or not can be determined from the ZREQbit in its entry of the fieldtable (see sec. 1.4.3). It is clear that at most one task in a virtual field can have access to the one-task-only part of that field.

The one-task-only part consists of locations 0-13 octal (containing 4 autoincrement registers) and locations 0150-0177 octal. When swapping in a task, nothing is stored in the one-task-only part. Therefore initially and after each SWAPOUT command the contents of these locations is undefined. For reasons that will become clear in section 1.7.6 no executable code must be stored in locations 0-13 octal.

The one-task-only part of virtual field 0 is used by the interrupt section and other parts of the monitor.

1.7.3 General constants

To save space in many tasks a number of commonly used constants, such as 2, 3, 77 and the like, are stored in page 0. They are listed in appendix A. One important constant on page 0 is a pointer to RPINTR. The IEXIT instruction used to return from a connected routine calls RPINTR via this pointer (see sec. 1.5.4).

1.7.4 Pseudo instructions

Page 0 holds a number of routines (or at least their entrypoints) which implement functions that cannot be executed by the task itself. Some routines have to access system tables and variables (group 1), others require task switch inhibition when applied in their full power (group 2).

Calling such a routine is termed executing a "pseudo instruction". To the programmer of the task these calls look like instructions that are executed, sometimes taking as or the contents of the next location as an argument. Like ordinary instructions pseudo instructions have a single word mnemonic.

Below we discuss the pseudo instructions (insofar this is not done elsewhere). For details we refer to appendix A. The routine implementing the pseudo instruction is indicated in the first line of each description.

Group 1.

MSREQ =JMS VMSNOD

No argument, but ac must be cleared at the call.

Request a message.

ac:=msptr (pointer to first informationword of message).

dfr:=ifr; link is disturbed.

MSFREE =JMS VMSNOD

ac:=msptr (pointer to first informationword of message).

Return a message.

ac:=contents of first informationword of message.

dfr:=ifr; link is disturbed.

VRCDF =JMS VVRCDF

Change datafield to requested field (see sec. 1.4.2).

ERHLT =JMS VERHLT

Halt this task. Not yet implemented.

VCDF =JMS VVCDF

Argument: next location bit 6-11 (6-bit fieldnumber).

Change datafield to indicated field.

VRDF =JMS VVRDF

No argument, but ac must be cleared at the call.

Read datafield.

ac:=virtual datafield.

dfr:=ifr.

CDFCUR =JMS VCDIF

Change datafield to instructionfield (see sec. 1.4.2).

Group 2.

The group 2 pseudo instructions are designed for the implementation of reentrant tasks (see sec. 2.2). Each incarnation of the task has its own tcb. Its variables and subroutine links are stored in an array, called the reentrancy array. BASE holds a pointer to that array. The group 2 pseudo instructions are meant to access data in the reentrancy array.

Note: of course nonreentrant tasks can benefit from the possibilities of these pseudo instructions.

All these pseudo instructions perform an indexed address operation. BASE is used as base register to which the relative address of the data to be accessed is added. This relative address is

usually indicated in the location after the pseudo instruction. At the conclusion of the pseudo instruction X holds the absolute address of the accessed data. This may be used for subsequent access of the same location.

Note, that the reentrancy array must reside in the instructionfield of the task.

GET =JMS VGET
ac must be cleared at the call.
Relative address in next location.
Fetch the contents of the addressed location.
dfr:=ifr.

PUT =JMS VPUT
ac holds data to be stored.
Relative address in next location.
Store data in ac in indicated location.
ac:=0.
 dfr:=ifr.

JUMS =JMS VJUMS
ac points at subroutine entry.
Relative address in subroutine entry.
Reentrant subroutine call. Return address is stored in the indicated location.
ac:=0.
 dfr:=ifr.

Example of subroutine call.

CLA	/ac:=0
TAD)TY	/get relocated address of subroutine
JUMS	/call
...	/will return here
TY, TYLINK	/subroutine entry holds relative address.
...	/subroutine body
GET; TYLINK	/fetch return address
DCA X	/temporary. untouched in task switch
JMP I X	/return.

This way of calling a subroutine makes the passing of arguments a bit complicated. Ac is used during the call operation proper and writing the arguments just behind the call does not work in reentrant code. We advise to pass arguments via the mq.

1.7.5 Entrance of the monitor command section

When a task executes the instruction CALL (=JMS VCALL), the monitor command section is entered. In fact a subroutine VCALL is called. VCALL is on the general page 0. It disables the interruptsystem and jumps to the label MONITOR in field 0. There the status of the task is saved temporarily, the specified command is fetched from the location after the CALL instruction and executed (see sec. 1.10).

1.7.6 Saving the pc

As stated in sec. 1.6.3 pseudo instructions (state C) and monitor commands (state D) are executed either with the interruptsystem disabled or with SDCINH set. Therefore a task should execute an IOF instruction before each pseudo instruction and the CALL instruction. We do not like tasks to touch the interruptsystem and hence devised the following solution.

The entrypoints of the VCALL routine and all routines that implement pseudo instructions are located at addresses ≤ 76 octal. The first instruction of these subroutines disables the interruptsystem. It will remain disabled until either the execution of taskcode is resumed, or the SCDINH flag is set. In this way dangerous interrupts can only occur just before the first instruction of such a subroutine, i.e. with pc ≤ 77 octal. As no other code is stored at addresses ≤ 77 octal (see sec. 1.7.2), the interrupt section may easily recognise this situation.

When the interrupt came just before the first instruction of the VCALL routine (monitor command section) the interrupt section acts as if the SCDINH flag had already been set. It resumes the program where it was interrupted, i.e. at VCALL+1. A possible schedule request will be treated at the end of the monitor command section.

If the interrupt came just before the first instruction of one of the other routines, the interrupt section fetches the pc from the entrypoint of the called routine. In this way the status of the task is saved as if it had not yet performed the pseudo instruction. This is effectively true as no instructions of the routine implementing it are executed yet. The pseudo instruction will be executed when the task is resumed.

These rules must of course not be applied when the usermode flipflop was set at the moment of the interrupt. The interrupted program then was not an ordinary task, but a user program running under timesharing control. Such programs do not use the general page 0, and are permitted to execute all kinds of code in their page 0.

When the interrupt section is about to enter the save status routine, it must perform the following check:

```
'if' usermode flipflop was set or pc >=100 octal then 'goto'
    save status 'fi';
'if' pc=VCALL+1 then resume interrupted program
'else' pc:=ifr[pc -1] -1 #value before pseudo instruction#
'fi'
save status: ...
```

1.7.7 Emergency escapes from pseudo instructions

Pseudo instructions are used so often, that the routines that implement them must be as efficient as possible. Although they could run very well with SCDINH set, setting and clearing this flag and the related testing of SCDREQ would take too much time. Generally they use a CIF instruction to inhibit interrupts until the next jump. Their code is all in line, and when the returnjump is executed the interruptsystem is automatically reenabled.

This construction does not allow testing of conditions. When such a test would fail and a jump were made to take proper action, this jump would reenable the interruptsystem prematurely with disastrous effects. The VCDF and VRCD pseudo instructions however require testing to ensure that the new datafield is in the actual core. The required jump is implemented by a JMS VCALL instruction. This instruction virtually enters the monitor command section. As may be seen in the previous section, a possible interrupt immediately after this JMS VCALL instruction will not disturb the program. The monitor command section may recognise such emergency calls from the value of pc at the call. It knows which pseudo instruction executed the emergency call and which check failed. It takes appropriate action.

1.7.8 Storing and updating the general page 0

A copy of the general page 0 is always present in field 0. Initially no other fields hold a copy of this page. As soon as tasks are swapped into a virtual field a copy of the general page 0 must be present in that field. Therefore the page allocator called by the task fetcher checks if the general page 0 is present in the field in which it allocates the task. If page 0 is not yet in that field, it is copied into it from field 0 (see sec. 1.8.3). The presence or absence of the general page 0 in a given field is indicated by the moninbit (bit 9) in the fieldtable (see sec. 1.4.3).

The copy of the general page 0 may be erased from a field if the field is given to a task for datastorage (REQFLD command). In that case the moninbit is cleared and the datastoragebit is set.

The copies of the general page 0 are not identical in all fields. Some CIF and CDF instructions in the general page 0 must agree with the actual field in which they are executed (e.g. the instructions to implement the CDFCUR pseudo instruction). So each time a virtual field holding a copy of the general page 0 is swapped into an actual field these instructions must be updated. This is done by the swaptask, unless the datastoragebit is set.

1.8. Storage allocation

There are three storage allocation systems in MC8. The first works on small nodes (maximum 16 consecutive locations) in field 0, the second allocates complete fields and the third system works on n consecutive pages of one field of the virtual core, where $1 \leq n \leq 31$.

1.8.1 Free core in field 0

During assembly all free core in field 0 is gathered in a singly linked list of nodes of consecutive free core. The first word of a node holds a pointer to the successor node (0 indicates the end). The second word points to the last location of the node. The list is organised from high to low, i.e. nodes corresponding to a higher address precede those corresponding to a lower address.

At the initialisation of MC8 this list is reversed and nodes corresponding to an address < 4000 octal are broken such that they consist of at most 15 locations. The latter is done to obtain a proper allocation of tcb's (see below). We will indicate this list as ahead, as AVHEAD is the location that points to its first node.

In field 0 4 avallists are used. Named after the locations pointing at their first nodes these are: avl2, avl3, avl5 and avl20 working on nodes of 2, 3, 5 and 16 words respectively. (16 = 20 octal.)

Nodes of length 2 are used for the timeoutqueue; nodes of length 3 are used in the free core chain; nodes of length 5 are used for message allocation; and nodes of length 16 are used to allocate tcb's. Initially the 4 avallists are empty.

When a routine in the monitor needs a node of certain length, it calls the appropriate getnode routine GN2, GN3, GN5 or GN20. The

called routine first inspects the corresponding avallist, and, if possible, takes a node from that list. If the avallist is empty ahead is searched for a node of sufficient length. From this node a node of the requested length is broken and the remainder, if its length is still ≥ 2 , is returned to ahead.

A node may be returned by calling the freenode routine FN, specifying the appropriate avallist as an argument. This routine adds the node to the corresponding avallist.

Remarks.

It is very difficult to make a useful static division of the free core into nodes of 2, 3, 5 and 16 locations. The number of nodes of each length that is needed depends on the way the system is used. In MC8 this division is postponed as long as possible and done dynamically. A disadvantage is, that the initial behaviour of the system has far more influence than the behaviour at later times.

As a consequence of the initialisation of ahead, nodes of length 16 (used to store tcb's) are always located at addresses ≥ 4000 octal, as was required (see sec. 1.1.2).

When at a given instant of time ahead is searched for a node but does not contain a suitable node, the system halts as if all free core in field 0 were exhausted. This need not be the case. Other avallists may still contain nodes, that, when merged with one another and the nodes in ahead, would yield sufficient free core. Such a garbage collection is not built in MC8.

Note: garbage collection would erase the extra influence of initial behaviour.

1.8.2 Requesting and returning fields

There are upto 64 virtual fields each of which is either free or occupied. Which of these two is the case is indicated in the fieldtable (see sec. 1.4.3).

When the page allocator needs a field it calls GETFLD. GETFLD searches the fieldtable for a free field. If no more fields are free, the system halts.

The page allocator returns a field by calling FRFLD.

A task may request a field for buffer storage. This is done by executing the REQFLD command. This command is implemented by calling GETFLD. The virtual fieldnumber is returned in the task's ac bit6-11. As page 0 of such a field may get overwritten, the moninbit in the fieldtable is cleared and the datastoragebit is set, to indicate that a copy of the general page 0 is no longer present in the field.

A task may return a field by executing the RTNFLD command, specifying the virtual fieldnumber of the returned field in ac bit6-11. To implement this command FRFLD is called.

1.8.3 The free core chain

To allocate tasks and buffers in the virtual core the "free core chain" (fcchain) is used. The fcchain is a singly linked, sorted list of 3-word nodes. Each node in the fcchain corresponds to a junk of 1-31 pages of free core in one field. If all 32 pages of a field are free, this is not denoted in the fcchain, but it is denoted in the corresponding entry of the fieldtable. The fcchain is sorted according to increasing field- and pagenumber.

Note: loosely speaking, we will often say that a junk is in the fcchain, meaning that the node corresponding to that junk is in the fcchain.

1-31 pages of free core may be requested from the page allocator. It returns a buffer of the requested length. The allocation of that buffer is indicated by specifying its virtual fieldnumber and the pagenumber of its first page. The buffer will never be located on page 0, because tasks may be allocated in the same field (perhaps this very buffer was meant for task storage). Page 0 is left free for a copy of the general page 0.

Two more options are specified to the page allocator. The buffer may be requested in a coreresident field and the buffer may be requested in a field where the one-task-only part is still free (this latter option is only used by the task fetcher (see sec. 1.9.4)).

The page allocator first searches a junk that is long enough (note that the fcchain is sorted), then checks the required field properties. If this check fails, searching is continued, otherwise the junk is either deleted from the fcchain (if it has exactly the right length) or the junk is broken and the remainder is left in the fcchain.

Algorithm of page allocator.

```
'INT' LREQ=#requested length#;
'BOOL' CRES=#buffer in coreresident field requested#,
'BOOL' OTO=#buffer in field with free one-task-only part
requested#;
'INT' F,L;
#initialise searching#;
S: #search junk of sufficient length#;
    F:=#fieldnumber of junk#,
    L:=#length of junk#;
'IF' 'NOT' (CRES=#cresbit 'of' fieldtable[F]#)
    'THEN' 'GOTO' S
'FI';
'IF' OTO 'AND' #zreqbit 'of' fieldtable[F]#
    'THEN' 'GOTO' S
'FI';
'IF' L=LREQ
    'THEN' #delete junk from fcchain#
    'ELSE' #brake junk#
'FI'
```

Note: each time the 'GOTO' S is executed, the search is continued where it was stopped.

If during the search the end of the fcchain is reached, no sufficiently long junk is present in the fcchain. GETFLD is called to fetch an entirely free field; the general page 0 is copied into it; page 0 is broken off; and a fresh junk of 31 pages (certainly long enough) is created.

A buffer may be returned to the page allocator by specifying its fieldnumber, the pagenumber of its first page, its length and field properties. The latter is used to clear the zreqbit in the fieldtable if the one-task-only part is free again.

The page allocator creates a node corresponding to the returned junk, possibly merges the junk with adjacent junks in the same field and if the, possibly merged, junk has less than 31 pages, inserts it in the fcchain. If the junk is 31 pages long, the whole field is free again (mind, that no stuff was allocated in page 0). This is only indicated in the corresponding entry of the fieldtable, not in the fcchain.

Layout of a node in the fcchain.

word	bit	meaning
0		Pointer to successor. 0000 terminates the chain.
1	0-5	Virtual fieldnumber of junk.
	7-11	Pagenumber of first page of junk.
2		Minus number of consecutive free pages.

Remarks.

Buffers not requested from the page allocator (but for instance unused pages of a field requested with the REQFLD command) must not be returned to the page allocator. The general page 0 may not be present in this field. When subsequently tasks are loaded into the returned buffer, serious errors can occur.

As the fcchain is sorted, there is a tendency to use the lower virtual fields first. This decreases the number of virtual fields in use, and hence the number of fieldswaps.

The way in which the coreresidentcondition is checked ensures, that as few fields are made coreresident as possible. This diminishes the risk of having many or all actual fields locked.

1.8.4 The REQPAG and RTNPAG command.

A task may request 1-31 consecutive pages of one field by executing the REQPAG command.

Ac must be cleared;

arg1 bit7-11=requested length;

arg1 bit 5=1: request buffer in coreresident field.

At return:

ac bit0-4 hold number of first page of buffer;

ac bit6-11 hold virtual fieldnumber of buffer.

To return a buffer, previously requested using the REQPAG command, a task may execute the RTNPAG command.

Ac bit0-4 =number of first page of buffer;

ac bit6-11 =virtual fieldnumber of buffer.

arg1 bit7-11 =length of returned buffer. ac:=0.

1.9 Monitor tasks

Monitor tasks are tasks that are built in and closely interrelated with the code of the monitor of MC8. They implement essential functions of the system. All monitor tasks (except the idletask) have priority 0 or run with SCDINH set, so that no other task can interrupt a monitor task when it is in execution. The idletask is the only task having priority 10 (octal). It is less important than all other tasks and each task may interrupt it.

1.9.1 The idletask

The algorithm of the scheduler is based upon the fact, that there is always a task that can be executed. Therefore the idletask has been built in the system. It does not perform pseudo instructions and monitor commands and its data- and instructionfield are both =00. So it is never waiting and runchecks cannot fail.

The idletask performs some innocent loop, that can be interrupted at each instant of time. Moreover, it clears the variable CURTSK, suggesting that the system is scheduling. This spares time needed for saving the status if another task becomes runnable (see sec. 1.6.2).

1.9.2 The system diskdriver

The system disk is a peripheral device on which virtual core and tasks are stored. A driver for this disk must be present in the system, as the system cannot swap in a drivertask when no diskdriver is available.

The system disk is divided into 8 logical units of 4096 blocks of 256 words each. Whether or not all units are (completely) available depends on the physical device used.

To each virtual field an area on logical unit 0 corresponds, a fieldslot. A fieldslot is 16 blocks (=4096 words) long. When a virtual field is not in the actual core, it is stored in its fieldslot.

Fieldslots are arranged one after another on logical unit 0. The blocknumber of the first fieldslot is determined by the value of VFBLOCK in the config file.

In the stl an 11-bit blocknumber can be specified for each task. This blocknumber indicates where on logical unit 0 the taskcode may be found. The actual blocknumber of the taskcode is computed by adding the value of TSBLOCK (specified in the config file) to the value given in stl.

The system diskdriver consists of two parts: a device-dependant section (the SDGO routine) and a device-independant section.

The device-dependant section accomplishes the actual datatransfer to or from the physical device on which the system disk is mounted. We shall not discuss it in this document. The listing in appendix A contains a version for the RK-8/E disk.

The device-independant section accepts, treats and reports messages. Two kinds of messages may be sent to the system diskdriver: normal messages and SWAPMS (the swap message).

Normal messages may be sent by any task. They specify in an obvious way a single datatransfer.

Layout of a normal message:

word	bit	meaning
0-1		Used by the system.
2	0	0: read, 1: write.
1-5		Number of pages to be transferred. 0 means 40 octal.
6-8		Actual fieldnumber of field of transfer. This seems a dangerous way to pass the fieldnumber, but it is safe if the DFPARM option is used (see sec. 1.2).
9-11		Logical unitnumber.
3		Core address of first word of transfer.
4		Blocknumber on disk.

When such a message is received, the SDGO routine is called to perform the actual datatransfer; the completion status is denoted in word 2 (the first informationword) of the message (0 indicates no errors); and the message is reported.

The system diskdriver recognises SWAPMS from its core address. It is allocated at a fixed core address and is only sent to the system diskdriver. Word 0 of this message has a fixed contents, such that it is always reported to the swaptask.

SWAPMS may be sent by the system upon failure of runcheck2 (see sec. 1.4.6). It specifies that a fieldswap must occur by passing the actual field of the swap, the virtual field to be swapped out and the virtual field to be swapped in to the system diskdriver.

Layout of SWAPMS:

word	bit	meaning
0		Tcbptr of swaptask.
1		Link in receive- or reportqueue.
2		Pointer in CORMAP, corresponding to the actual field.
3		Pointer in the fieldtable, corresponding to the virtual field to be swapped out.
4		Pointer in the fieldtable, corresponding to virtual field to be swapped in.

When this message is received, SDGO is called twice, once for the transfer of each virtual field. A possible error in the transfer halts the system. At the completion of the transfer, the message is reported.

Note the following optimisation. When the fieldtable indicates that the virtual field to be swapped does not contain data, SDGO is not called, so that no actual transfer takes place.

Remark.

As stated in sec. 1.2, the best way to pass actual fieldnumbers with the DFPARM option is, to lock the datafield before passing its

actual fieldnumber. This ensures that the designated field is still present in that actual field when the receiver accesses it. Locking is unnecessary when sending a message to the system diskdriver. The field under concern cannot be swapped out meanwhile, because a possible message to the system diskdriver to do so, will be behind the current one in the receivequeue, and hence be treated later.

1.9.3 The swaptask

The swaptask does some bookkeeping in system tables after a fieldswap. It is activated by reporting SWAPMS (see previous section). For a description of the swaptask we refer to sec. 1.4.6.

1.9.4 The taskfetcher

Runnable tasks, the code of which is still on disk, are swapped into the virtual core by the taskfetcher. Such a task must be built in the system, as if it were on disk it could not swap itself in.

When the scheduler has selected a runnable task, it checks the ondisk-bit (runcheck1, see sec. 1.3.3). If this bit is set, the task is removed from the runqueue and a message is sent to the taskfetcher to swap in this task.

Loop of the taskfetcher.

1. Wait for a message, specifying that some task must be swapped into the virtual core.
From the information in the message a pointer to the task's tcb is computed. Using tcb[6] (static tasknumber) the information in the stl may also be accessed.
2. Fetch length and fieldproperties (task uses one-task-only part, task runs in coreresident field) of the task from the stl.
3. Request an appropriate buffer from the page allocator.
Note, that the page allocator guarantees that a copy of the general page 0 is present in the field of the buffer.
4. Send a message to the system diskdriver to read the code of the task into core. The blocknumber is computed from tcb[15].
5. Update pointers in the code of the task: relocation (see sec. 1.1.1).
6. Update the tcb of the task.
7. Reinsert the task in its runqueue.
8. Return to step 1.

The following locations in the tcb are updated:

- tcb[7] Datafield and instructionfield are both set to the field in which the task is stored.
- tcb[9] If no value (0000) was indicated in tcb[9], the pc, this is considered as the start of the task. Pc is made to point to the first location after the first relocatable pointer sequence (see sec. 1.1.1). That is, if no value was indicated for pc, the task is started at the first executable instruction on its first page.
If a value was present in tcb[9], this value is regarded as relative to the task-offset. Hence pc is then updated by adding the task-offset.
- tcb[14] The new first page of the task is stored in bit0-4 of this word.

When a task executes one of the wait commands STALL, WTRP, SNDWTR or WTMS, it may specify the SWAPOUT option. If the task is actually set to wait (which need not be the case), the pages occupied by its code are returned to the page allocator. As soon as the task is runnable again, the taskfetcher swaps a fresh copy of its code back into core.

Specifying the SWAPOUT option is useful only if the task expects to be waiting for quite a long time (typically more than 0.5 sec). By long we mean long when compared to the time required to swap in the taskcode, which is about 0.1 sec.

Moreover, one must be aware of the side effects of the SWAPOUT option:

- a fresh copy of the code is swapped in, so all information stored in the code is lost;
- the datafield becomes equal to the instructionfield; if other tasks have pointers pointing into the code of the task under concern (which they shouldn't), these pointers become meaningless.

Remark.

When the taskfetcher updates the code of a task, it accesses a field that perhaps is not in the actual core. Hence runcheck2 may fail when the taskfetcher is selected. To avoid the danger for deadlocks, the taskfetcher does not have priority 0, but priority 1. As it uses code of the monitor, it must run with SCDINH set. The danger for deadlock is evident from the swap algorithm explained in sec. 1.4.6.

1.9.5 The timer

The timer keeps track of the system time and treats the "timeoutqueue". It is driven by clock-interrupts and runs once each 0.1 sec.

When TIMEMS (a special message) is reported, the timer starts to work. It increments the system time: a 2-word counter that counts the time modulo 2^{24} in units of 0.1 sec. Next the timer treats the timeoutqueue (see sec. 1.9.6) and resets itself to wait for the next report of TIMEMS.

How precisely TIMEMS is reported depends on the implementation. In the version listed in appendix A, a clock is used, causing interrupts each 0.1 sec. The corresponding intslot is claimed and TIMEMS is attached to it.

1.9.6 The timeoutqueue (toq)

When a task executes the STALL command it sets itself to wait for some distinct period. In fact it delays its execution for a while, therefore such a task is termed "a task in delay". The length of the delay is determined from the so-called "timeout value" specified in ac at the execution of the STALL command. Each 0.1 sec this timeout value is incremented by 1 and on overflow the task is reinserted in its runqueue. When for instance 7771 octal is specified in ac, the task is runnable again after 0.7 sec.

Note, that ac=0 when the execution of the task is restarted.

To implement this command the "timeoutqueue" (toq) is used. Upon execution of the STALL command a node corresponding to the task is inserted in the toq. The toq is a singly linked list of 2-word nodes. The first word of a node points to its successor (0000 terminates the queue). The second word of a node points to tcb[8], ac, of the task in delay. Each time the timer treats the toq, it uses the latter pointer to increment the task's ac. When overflow occurs the task is reinserted in its runqueue and the node is deleted from the toq. Note, that a pointer to tcb[8] cannot be 0000.

When a task executes one of the commands WTRP, SNDWTR or WTMS, a task may specify the TIMEOUT option. If the task is actually set to wait (which need not occur), a node is inserted in the toq just as if a STALL command had been executed. Of course the other waitconditions specified in the command are also set.

As long as the task is waiting, the timeout value in ac is incremented each 0.1 sec. If overflow occurs the task is reinserted in its runqueue with ac=0 and the other waitconditions are ignored. If one of the other waitconditions expires (a message is sent or

reported to the task) before the end of the delay, the msptr is set into ac, as usual, and the task is reinserted in its runqueue. In the latter case the TIMEOUT option had no effect.

So the use of the TIMEOUT option ensures that the task waits only for a limited time (determined by the timeout value in ac). Either the specified waitcondition expires within that period, in which case task execution is restarted with a msptr in ac, or after the specified period task execution is restarted with ac=0.

Implementing the TIMEOUT option was difficult. Tasks in delay have a node in the toq, pointing at their ac. Each time the timer runs, it increments this ac. When the waitcondition expires, a msptr is loaded into the ac. From that moment it is no longer allowed to touch the task's ac.

Removing the node from the toq takes too long, as the toq is singly linked. Therefore tcb[5] of a task in delay points to its node in the toq. When the task is reinserted in its runqueue, this pointer is used to clear word 2 (the pointer to ac) of the node. The timer recognises such a node and removes it from the toq without taking further action.

Note, that tcb[5] of a task in delay is not used to point in a runqueue as such a task is not runnable.

1.10 Monitor commands

The monitor provides a number of helpful functions for a task. To request for such help a task must execute a CALL instruction. Thereafter the commandword is denoted, followed by 0, 1 or 2 arguments, depending on the command given. Sometimes the ac of the task is also used as an argument. In fact a task "commands" the monitor to do something, therefore this is termed "executing a monitor command".

The command is specified in the commandword. In addition some commands allow options to be specified in the commandword. Layout of the commandword

bit meaning

0 Check only option.

This option changes the WTMS and WTRP commands into CHKRCQ and CHKRPO respectively (see sec. 1.2).

1 NONREP option (see sec. 1.2).

2 KEEP option (see sec. 1.2.1).

3 SWAPOUT option (see sec. 1.9.4).

4 TIMEOUT option (see sec. 1.9.6).

5 DFPARM option (see sec. 1.2).

6-10 Command specifier.

11 1: arguments present; 0: no arguments present.

When the CALL (=JMS VCALL) instruction is executed, the routine VCALL on the general page 0 is called. This routine disables the interruptsystem and jumps to the label MONITOR in field 0. There the status of the calling task is saved temporarily, the SCDINH flag is set and the interruptsystem is reenabled. Now the monitor is ready to treat the call.

First remember that one of the routines on the general page 0 may have executed an emergency call. This may be derived from the value of the saved pc. If the pc points in one of the page 0 routines, an emergency call was indeed executed. The system retrieves the actual status of the task from the page 0 routine under concern, saves it temporarily and then takes further action depending on which routine called.

If the call was no emergency call, but a normal call executed by a task, the designated action is indicated in the commandword. The commandword and possible arguments are fetched from the field of the caller (thereby incrementing the value of the saved pc) and the system jumps to the appropriate routine, using bit6-10 of the commandword as a switch.

The effect of monitor commands is described in detail in other sections of this paper. Appendix E summarises all monitor commands and their options and refers to the sections in which they are described.

2 Miscellaneous remarks

2.1 Creating tasks

Occasionally a task may wish to create other tasks, e.g., to load a task from a device other than the system disk or to create a new incarnation of a reentrant task (see sec. 2.2). Such a newly created task is termed a descendant of the task that created it.

The code of the descendant must be provided somewhere in the virtual core, maybe in a buffer acquired using the REQ-PAG command or as part of the code of the task that created it. A static tasknumber and a task controlblock must be attached to the descendant. To create tasks and to remove created tasks the monitor commands discussed below have been added.

REQSTL.

Request entry in stl (request st#).

Arg1, arg2: contents of entry in stl.

ac:=st#.

Note, that a negative contents of arg1 is interpreted as tcbptr.

REQTCB.

Attach a tcb to, or retrieve the tcb of a task.

ac =st#.

ac:=tcbptr (pointer to tcb[5]).

If no tcb is attached yet to the specified entry in the stl (word 0 >0), a tcb is requested, prefilled with initial values (see sec. 1.1.3), and its tcbptr denoted in word 0 of the entry in the stl.

The tcbptr (now) present in word 0 of the entry in stl is returned in ac.

FILTTCB.

Fill the tcb of a task.

ac =st#.

Arg1 =pointer to value array.

First attach a tcb to the task (or retrieve its tcb). Then overwrite the first 15 locations of the tcb with the values indicated in the value array. Arg1 points at word 0 of the value array.

Note: tcb[15] is not overwritten. This location holds the original contents of word 0 of the entry in stl.

RTNTCB.

Return a tcb.

ac =st#.

The tcb is detached from the specified entry in the stl and returned to the avallist system. Word 0 of the entry is overwritten with the contents of tcb[15].

The request and fill commands above allow tasks to assign a st# and a (properly filled) tcb to a descendant. The RTNTCB command may be used to return a requested tcb.

To return the entry in the stl used by a descendant that ceases to exist, the task must explicitly clear word 0 of that entry. A pointer to that word is computed by taking twice the st# and adding the constant ZSTL (general constant).

Summary.

Common scheme for creating a descendant:

1. Provide code.
2. Request st#, using REQSTL.
3. Request and fill tcb, using FILTCB.
4. Send message using the st#.

Common scheme for erasing a descendant:

1. Stop descendant if it is still runnable, using STOP.
2. Return its tcb, using RTNTCB.
3. Clear corresponding entry in stl.
4. If possible return the buffer in which the code was stored.

2.2 Reentrant tasks

A reentrant task is a task that operates on "reentrant code", i.e., a task that executes code that simultaneously may be executed by other tasks. All tasks that operate on the same code are termed incarnations of the reentrant task. Such incarnations have different tcb's, and thus may have different values in their registers.

By nature the code of PDP-8 computers does not allow reentrancy. Simple tools like a hardware stack and hardware indexed addressing are missing. Moreover, the return address of a subroutine is stored in a fixed location each time the subroutine is called.

In MC8 the lack of hardware indexed addressing is remedied by the GET and PUT pseudo instructions (see sec. 1.7.4). These pseudo instructions use the register BASE as offset of the address and the location after the instruction as an index. As the value of BASE may differ from one incarnation to the other, each incarnation can store its variables in its "own" array. This array is termed the "reentrancy array".

The JUMS pseudo instruction (see sec. 1.7.4) may be used to store subroutine links in the reentrancy array.

If an incarnation would only store variables in its registers and its reentrancy array, its code would remain unchanged and hence there would be no special problems with reentrant tasks. This however is clumsy and time consuming. The programmer is continuously tempted to store some temporary value in some "local" variable. This would not be so bad, if not at each moment the execution of an incarnation could be interrupted and the execution of another incarnation of the same task be resumed. This latter incarnation might disturb the variable, before the former one is resumed.

Things would be much better if incarnations of a given reentrant task could only be interrupted by other incarnations of the same task at fixed places in the code. At such places the programmer could take care that all values to be used later are properly stored in registers or the reentrancy array, which are inaccessible from other incarnations. There are two methods to achieve this in MC8.

Reentrancybit (tcb[6] bit1).

The use of the reentrancybit requires that all incarnations of a reentrant task have the same priority and that they all have the reentrancybit set. This need not be the case in MC8, but usually it is.

When the scheduler selects a task, whose data- and instructionfield are not both in the actual core, it normally continues inspecting the runqueue of the same priority (see sec. 1.3.3). The scheduler will skip over this runqueue and start inspecting the next important runqueue if the selected task has its reentrancybit set.

This has the following effect.

The execution of an incarnation is started only if it is the first incarnation in the runqueue. Other incarnations of the same task cannot be executed until the current incarnation has been removed from the runqueue, i.e., until it has executed a "wait" command (STALL, WTRP, WTMS SNDWTR).

For instance let C1 and C2 be incarnations of a reentrant task, and let C1 precede C2 in the runqueue. The scheduler will always select C1 first, and either start up its execution or skip over the rest of the runqueue, including C2. Hence C2 will not be executed as long as C1 is present in the runqueue.

Using this method the only delicate places in a reentrant task are those places where wait commands (see above) are executed. A severe disadvantage is, that incarnations of other reentrant tasks (and possibly lots of other tasks) behind the first incarnation in the runqueue are not started until the first incarnation sets itself

to wait, even if their fields are in the actual core. They all have to wait until the fields of the first incarnation have been swapped in.

The `scdinhbit` (`tcb[11] bit3`).

Tasks that have this bit set run with the `SCDINH` flag set. This prevents them from being interrupted by any other task, including other incarnations of the same reentrant task. Other tasks can only interfere when a monitor command is executed, or when the task tries to access a field that is not in the actual core.

Delicate places are those places where monitor commands are executed and places where a `VCDF` and a `VRCDF` is executed. The danger for the `VRCDF` instruction is restricted to the first call after a monitor command; later calls will find the requested field in the actual core.

A disadvantage of this method is, that it overrides the normal priority scheme. A task running with the `scdinhbit` set must provide enough monitor calls in order not to hamper normal progress of other tasks.

Remark.

A reentrant task must never execute the `EXIT` command or use the `SWAPOUT` option. This would erase the code from core, although other incarnations might still be using it.

2.3 Protect tasks

Protect tasks are tasks running with the `usermode` flipflop set. They constitute the timesharing mode of `MC8`. Although we have no definite ideas about the implementation of such tasks, a few words must be spent on this subject. Otherwise some parts of the code of `MC8` would look very strange.

A protect task is meant to simulate a dedicated installation to a user. When the protect task runs, the program stored in the memory of the simulated machine is executed. In order to do so, the fields of the virtual core in which the simulated memory is stored are swapped into the actual core (for so far they are accessed), and the program is started at the location and field indicated in the `tcb`. Note however, that the `usermode` flipflop is set.

When the simulated program tries to execute an instruction that might effect other parts of the system or perhaps the outside world, a "trap" occurs. That is, the protect task is interrupted hardwarewise, and when subsequently the deviceflags in the `skipchain` are interrogated (see sec. 1.5.2), the `sint`-instruction (6254) will

skip to indicate that a trap occurred. Some emulator task (or perhaps a connected routine) may now be started to interpret the trapped instruction and to take appropriate action.

So all interaction between the outside world and the simulated program and all field access of the simulated program (note, that CDF and CIF instructions and the like also cause traps) is controlled by the emulator task.

The need for controlling the field access is shown in the following example.

Assume the simulated program attempts to execute a CDF 10 instruction in order to access simulated field 1. The emulator task now must find out in which virtual field field 1 of the simulated memory is stored, check if this virtual field is in the actual core and, if so, make the datafieldregister point to the correct actual field.

Before the simulation is started, the user must specify to the emulator task which configuration he wants to be simulated (e.g. the number of memoryfields and the type of peripheral devices). The emulator task requests buffers to store the simulated memory, using the REQFLD command. A simulated program of course has full access to the fields of its memory. Hence it is impossible to load a general page 0 in such fields. As a consequence the pc of a protect task may point into page 0 without special implications and registers such as BASE and X do not exist. To facilitate the treatment of such peculiarities the routine CDFUF is patched. Normally this routine gives the monitor access to the field in which the taskcode is stored. For protect tasks this routine is changed such that it skips. This is done during task start up and is used in the save status routine.

Note, that protect tasks cannot execute monitor commands or pseudo instructions.

2.4 The task library

The structure of the task library has little effect on the code of MC8. In the code of MC8 a few tasks, the monitor tasks, are incorporated. Other tasks must somehow be loaded into the system when they are required and unloaded afterwards. This must be done by a task, called the "taskloader". The structure of the task library does effect the taskloader, but not the code of MC8.

The tools available to the taskloader were discussed in sec. 2.1. In general it is sufficient to assign a st# to the task to be loaded and to specify the contents of its entry in the stl, including a blocknumber on disk. If this code is to be swapped in by the taskfetcher, the taskcode must reside on a special area of unit 0 of the system disk (see sec. 1.9.2).

A problem arises when a task wants to communicate with other tasks. It must know the st# of the task it wants to send a message to. The st# is determined dynamically and hence cannot be assembled in the code of a task. The only statically known identification of a task is its taskname. The taskloader assigns a st# to a loaded task and hence it is the job of the taskloader to replace tasknames by st#'s.

This can be done in various ways. It heavily depends on the structure of the task library and the way in which tasknames are indicated in the code. The solution we chose is briefly outlined below. For some applications it may prove successful, in other cases it may be awkward and other methods must be used. In appendix C our version of the taskloader is listed.

2.4.1 Families of tasks

In our task library tasks are grouped in families. The taskloader loads (or unloads) complete families of tasks. A taskname consists of two parts: the familyname and the membername. There is one special family named GLOBAL. The members of GLOBAL are very important tasks, including the monitor tasks.

A task can communicate to members of its own family, and to members of GLOBAL. To specify tasknames at assembly time we added three pseudo operands to the assembler (see [5]), namely:

TASKNAME	To specify the taskname of the task that is assembled;
GLTASK	To indicate a member of GLOBAL;
LCTASK	To indicate a member of the family of the task that is assembled.

In the task library each task is preceded by a so-called task info block. This block holds a list of tasknames used by the task, together with pointers indicating where in the code the corresponding st#'s must be inserted.

During the initialisation the taskloader loads GLOBAL. Other families are loaded or unloaded when appropriate messages are received. GLOBAL remains always loaded.

To load a family of tasks the taskloader first assigns a st# to each member of the family. Next it updates the code of each task using the task info block. Note, that in order to do so, the code of each task must be swapped in, updated and swapped out again, even for those members that are not actually used.

2.5 Configuring MC8.

MC8 can be used on a variety of PDP-8/I and PDP-8/E installations.

Minimum requirements are: 16k memory, a disk and a clock.

The major part of the code of MC8 is equal on all installations. Some sections may vary slightly and thus are enclosed in the angular brackets "<...>" of conditional assembly.

A number of parameters specifying the configuration are grouped in the config file. They either set values (such as ACTMAX) or control conditional assembly (for instance EAE).

The following must be specified in the config file: (Values indicated with H are prescribed by the hardware used. Values indicated with U are chosen by the user.)

1.H Processor type.

MC8 running on PDP-8/I or PDP-8/E, EAE (Extended Arithmetic Element) present or not.

2.H Size of the actual core memory.

3.U Size of the virtual core memory.

4.U The length of the static tasklist.

This defines the maximum number of tasks that can simultaneously be loaded in the system.

5.HU The devices known in the skipchain and their order.

6.U Assembly parameters for extra error check and statistics, if desired (see sec. 2.7).

7.U The devices on which the system disk and the system clock are to be implemented.

8.U The layout of logical unit 0 of the system disk.

The device-dependant parts of MC8 are assembled conditionally, controlled by parameters in the config file. If devices are to be used for which no code is yet available in MC8, the following must be added.

If the device can cause interrupts an initial intslot must be specified in the skipchain.

If the device is to be used as system disk, a device-dependant version of the SDGO routine must be provided.

If the device is to be used as the system clock, things must be arranged such that TIMEMS is reported each 0.1 sec. This sets the timertask to work. the timertask itself is fully device-independant.

Initialisation.

At the start of MC8 an initialisation routine is called. This routine activates the scheduler when it is terminated.

The first task scheduled is MIINIT, an initialisation task built in

the system. MTINIT ends by erasing all initialisation code from core, and by clearing its entry in the static tasklist.

The initialisation routine organises the avallist systems, initialises all devices and starts the clock. MTINIT can be used to do config-dependant initialisation that requires the use of monitor commands. Amongst others the taskloader may now be brought into the system, and it can be indicated in the task library that monitor tasks, such as the system diskdriver, are permanently loaded in the system.

The initialisation routine and MTINIT may be assembled separately. Their code must be loaded together with the code of MC8.

2.6 Error handling

The error handling in MC8 is very poor, due to the trade-off of constructing a compact and small system.

At present the errors that are detected generate an error message on the console teletype and halt the system. The error message specifies the address in the monitor where the error was detected.

If the parameter CHECK is defined in the config file, some extra error checking is performed.

Several types of errors can be distinguished:

Resource exhausted.

For instance a buffer can be requested while the virtual memory is completely in use, or a node in field 0 can be requested while the avallist and ahead are empty. Errors of this type can sometimes be recovered by garbage collection. Another improvement would be to issue some warning when a resource runs to its end.

Hardware errors.

Sometimes hardware errors may be detected, for instance errors in a disk transfer. Such errors are unrecoverable. The best that can be done, is to halt the system in such a state, that important data may easily be retrieved.

Software errors.

Checking arguments and registers at the execution of each monitor command would detect most software errors in a task. Such an erroneous task could be stopped immediately in order to limit its effect on other parts of the system.

One of the problems of error handling in MC8 is, that it is difficult to stop a task that is in error or that is about to exhaust

some resource. Of course the task can be stopped, but it is impossible to retrieve which buffers and messages must be returned.

MC8 error handling can be improved at the cost of 1 extra field of coreresident code.

2.7 The system bulletin

To facilitate debugging of tasks and investigating the performance of the system, a special program, MC8BUL, was developed. MC8BUL produces a document, the system bulletin, in which the contents of important variables of MC8 is listed.

When the parameter STATX was defined in the config file, at the cost of some time and space a lot of counting is done while MC8 is running. The data so gathered are also listed in the system bulletin. They are meant to give insight in the performance of the system.

The present version of MC8BUL runs only under control of the OS/8 operating system. To use it, one must save the contents of field 0 after a run of MC8 on a file. This file may subsequently be fed to MC8BUL.

MC8BUL first calls the commanddecoder to find out where its input is to be found and where its output has to go. Moreover, it accepts two options: /Z, to indicate that the runcount must be reset, and /Y, to indicate that the contents of system variables must be dumped (normally only the result of the counting is listed).

Assuming that /Y was specified and STATX defined in the config file, MC8BUL gives the following output:

1. Date and runcount.
The runcount is incremented each time MC8BUL is run and may be reset by specifying the /Z option in reply to the commanddecoder.
2. A number of system variables, specifying the current task, the status of the SCDINH and SCDREQ flags, the last status saved in the interrupt section and the monitor command section.
3. The contents of the runqueues.
4. The skipchain.
5. The coremap.
6. The timeoutqueue.
7. The contents of the fieldtable.
8. The free core chain (see sec. 1.8.3).
9. The number of nodes present in the avaiillists.
10. The status of all tasks loaded in the system.
11. The number of unused entries in the static tasklist. Note, that this value gives the number of entries that still were available in the stl at the peak of its occupation.
12. The number of hardware interrupts seen.

13. The number of schedules at the end of the interrupt section and after a monitor command. Note, that schedules interrupting the idle loop are not counted.
14. The peak length of the messagequeues of the tasks.
15. The number of ordinary swaps to or from the system disk, the number of fieldswaps and the number of swap errors.
16. The number of virtual fields that was actually accessed.
17. The number of monitor commands executed.
18. The amount of space that was used by the avallist systems in field 0.
19. The run time, the amount of time spent in the idle loop and the time inactively waited for fieldswaps.

Items 12-19 are only listed if STATX was defined in the config file. To specify the status of a task, the following values are output:

- Its name, retrieved from the task library.
- The contents of its entry in the static tasklist.

If a tcb was attached:

- The contents of tcb[6-15] (see sec. 1.3.3).
- The contents of the claim word (tcb[0]) and the first 10 messages in the receivequeue.
- The report for which the task is waiting (tcb[2]), if it is, and the first 10 messages in the reportqueue.
- The contents of the wait word (tcb[4]) and the linkword (tcb[5]). For their use see sec. 1.3.

In appendix D a system bulletin is listed.

3 Literature

- [1] Hagen P.J.W.ten, Vliet R.van, Tekstverwerking,
MC Rapport IN 11/76
- [2] Anthony J.F. A real-time Foreground/Timesharing Background
Operating System for a minicomputer.
TH-Delft PHD Thesis 1975
- [3] RTS-8 Users Manual
DEC-O8-ORTMA-B-D
1973
- [4] OS-8 Handbook
DEC-S8-OSHBA-A-D
- [5] PDP DOC Series LD Series
Last edition
- [6] Tanenbaum A.S, Structured Computer Organisation
CH5.5 Prentice Hall 1976
- [7] Brouwer A.E, Vliet R.van, Wakker W,
PDP8 Simulation
MC Rapport IW 74/77

```

1          /APP. A0.  CONFIGURATION FILE,
2          /PROCESSOR TYPE,
3          0001  PDP8E=1
4          /PDP8I=1
5          0001  EAE=1
6
7          0001  IFDEF EAE <HARDMQ=1>
8          0001  IFDEF PDP8E <HARDMQ=1>
9          7701  IFDEF  HARDMQ  <GETMQ=CLA MQA; STORMQ=MQL>
10         7421
11         IFNDEF HARDMQ  <GETMQ=TAD MQ; STORMQ=DCA MQ>
12
13         /INDICATE ACTUAL CORESIZE (HIGHEST ACTUAL FIELD)
14         0007  ACTMAX=7
15
16         /INDICATE VIRTUAL CORESIZE (HIGHEST VIRTUAL FIELD)
17         0077  VIRMAX=77          /MUST BE <=77
18
19         /INDICATE LENGTH OF STATIC TASK LIST (HIGHEST ST#)
20         /THIS VALUE DETERMINES THE MAX NUMBER OF TASKS THAT
21         /CAN RUN SIMULTANEOUSLY.
22         0077  MAXSTL=77          /MUST BE <=177
23
24         /CONFIGURATE SKIPCHAIN.
25         /INDICATE YOUR DEVICES IN THE ORDER OF THEIR
26         /INTERRUPT PRIORITY.
27         NOPUNCH          /IDEA OF MULTI 8.
28         0000  *0
29         INTDEF,
30         00000 0000  RK8E,  0
31         00001 0000  KM8E,  0
32         /MULTIP,          0          /MULTIPLEXER CLOCK,
33         00002 0000  PDP8IE, 0;0      /TWO DEVICES !!!!!!!!!!!
34         00003 0000
35         /PDP11, 0;0
36         /RF08,  0
37         00004 0000  DK8EP,  0
38         00005 0000  TT1,    0
39         00006 0000  KB1,    0
40         00007 0000  TT2,    0
41         00010 0000  KB2,    0
42         00011 0000  TT3,    0
43         00012 0000  KB3,    0
44         MAXDEV,
45         ENPUNCH
46         0370  IFDEF TT2 <HCT=6420-6030>
47         0430  IFDEF TT3 <HP=6460-6030>
48
49         6041  IFDEF TT1 <TSF1=TSF;TCF1=TCF;TPC1=TPC;TLS1=TLS>
50         6042
51         6044
52         6046
53         6031  IFDEF KB1 <KSF1=KSF;KCC1=KCC;KRS1=KRS;KRB1=KRB>
54         6032
55         6034

```

```
56          6036
57
58          6431  IFDEF TT2 <TSF2=TSF+HCT;TCF2=TCF+HCT;TPC2=TPC+HCT;TLS2=TLS+HCT>
59          6432
60          6434
61          6436
62          6421  IFDEF KB2 <KSF2=KSF+HCT;KCC2=KCC+HCT;KRS2=KRS+HCT;KRB2=KRB+HCT>
63          6422
64          6424
65          6426
66
67          6471  IFDEF TT3 <TSF3=TSF+HP;TCF3=TCF+HP;TPC3=TPC+HP;TLS3=TLS+HP>
68          6472
69          6474
70          6476
71          6461  IFDEF KB3 <KSF3=KSF+HP;KCC3=KCC+HP;KRS3=KRS+HP;KRB3=KRB+HP>
72          6462
73          6464
74          6466
75
76          /SYSTEM STATISTICS WANTED?
77          0001  STATX=1
78          /EXTRA ERROR CHECK WANTED?
79          0001  CHECK=1
80
81          IFDEF STATX <
82          /SET VALUE FOR MEASURING TIME IN IDLELOOP,
83          DECIMAL
84          6331  IFDEF PDP8E < TCKLEN=3289          /ABOUT 100.000/30.4 FOR PDP8/E>
85          IFDEF PDP8I < TCKLEN=2777          /ABOUT 100.000/36 FOR PDP8/I>
86          /USED FOR IDLETIME STATISTICS,
87          OCTAL
88          >
89
90          /SELECT SYSTEM DISK.
91          0000  SYDISK=RK8E
92          /SYDISK=RF08
93          /SELECT SYSTEM CLOCK.
94          0004  SYTIME=DK8EP
95          /SYTIME=MULTIP
96
97          /CONFIGURE LOGICAL UNIT 0 OF SYSTEM DISK.
98          4000  VFBLOK=4000 /START BLOCK OF VFLODSLOTS,
99          IFNZRO VFBLOK^17 <ASS ERR! VFBLOK MUST BE MULTIPLE OF 20>
100          6000  TSBLOK=VIRMAX+1^20+VFBLOK /STARTING BLOK OF TASK LIBRARY,
101
102          /SOME STUFF FOR PROGRAMS USING THE TASKLIBRARY.
103          /NOT REQUIRED FOR MC8!!!!!!!!!!!!!!
104          /ASSUME DIRECTORY SWAPPED IN; TSBLOK+1 STARTS AT LOC200.
105          0200  DCDIR=200
106          0400  NRT=DCDIR+200
107          5000  FDCMAX=5000          /FIRST FREE LOC AFTER DIRECTORIES.
```

/APP, A1. MC8.1. PAGE 0, INTR SECTION PAL8-V9H 02/14/77 PAGE 3

```
108      /APP. A1. MC8.1. PAGE 0, INTR SECTION, SCHEDULER.
109
110
111      /***** M C 8 *****/
112      /MULTICORE 8.
113      /R VAN VLIET MATH' CENTR' A'DAM.
114      /DATE 2 DEC 1975.
115
116      /*****
117      /
118      /OPERATING SYSTEM FOR D.E.C. PDP8.
119      /
120      /A PDP8 HAVING UPTO 64 VIRTUAL MEMORYFIELDS IS IMPLEMENTED.
121      /THE SYSTEM REQUIRES AT LEAST 16K MEMORY AND A REASONABLY
122      /FAST DISK.
123      /
124      /*****
125
```

```

126          /CONVENTIONS AND ABBREVIATIONS.
127
128          /      /!      DEAF.
129          /      /      SECTIONS ACCESSIBLE ONLY FROM STATE C OR
130          /      /      FROM STATE D.
131          /      /\      CRITICAL SECTION OF STATE B ACCESSING
132          /      /      VARIABLES COMMON WITH STATE D; DEAF.
133          /      /!\     COMMON SBR OF STATES B AND D; DEAF.
134          /      /-      STATE CHANGING SECTION; DEAF.
135          /      /      NOTE: SOME ROUTINES ARE ACCESSED BY STATE C
136          /      /      (DEAF) AND B (SOMETIMES DEAF). THIS CANNOT CAUSE
137          /      /      TROUBLES AS THESE STATES DONT INTERRUPT EACH
138          /      /      OTHER. THESE ROUTINES ARE NOT ALLOWED TO TOUCH TH
E
139          /      /      INTERRUPT SYSTEM. THEY ARE NOT MARKED!!!
140          /      [N]     ARRAY INDEX.
141          /      /      ARRAY INDICES COUNT FROM 0.
142          /      /      ARR[N] INDICATES THE NTH ENTRY IN ARRAY ARR.
143          /      [N,M]   INDEX IN 2-DIMENSIONAL ARRAY.
144          /      M (N)   CYCLES.
145          /      /      THE LENGTH OF SOME ROUTINES IS INDICATED IN
146          /      /      MEMORYCYLCES.
147          /      /      M REFERS TO PDP8/E, N REFERS TO PDP8/I.
148          /      ASSXN   HELP REGISTERS ASSX1, ASSX2... ARE USED DURING
149          /      /      ASSEMBLY.
150
151          /      TSK      TASK
152          /      TSKSW    TASKSWITCHING
153          /      Q        QUEUE
154          /      CHN      CHAIN
155          /      TAB      TABLE
156          /      AC       ACCUMULATOR
157          /      IF       INSTRUCTIONFILED
158          /      DF       DATAFIELD
159          /      L        LINK
160          /      FL       FLAGS. PROCESSOR STATUS (PDP8E).
161          /      MQ       MULTIPLIER QUOTIENT. A REGISTER.
162          /      EAE      EXTENDED ARITHMETIC ELEMENT (PROCESSOR OPTION).
163          /      SC       STEPCOUNT.
164          /      INTR     INTERRUPT
165          /      INT      INTERRUPT
166          /      LOC      LOCATION
167          /      W        WORD
168          /      PT       POINT
169          /      CT       COUNT
170          /      FLD      FIELD
171          /      VFLO     VIRTUAL FIELD
172          /      MON      MONITOR
173          /      ARG      ARGUMNENT
174          /      SBR      SUBROUTINE.
175          /      PARAM    PARAMETER
176          /      OPT      OPTION
177          /      FUNC     FUNCTION
178          /      MS       MESSAGE
179          /      MSS      MESSAGES
180          /      RP       REPORT

```

/APP. A1. MC8.1. PAGE 0, INTR SECTION PAL8-V9H 02/14/77 PAGE 4-1

181	/	RCQ	RECEIVE QUEUE (MSS SENT TO A TSK GO THERE)
182	/	RPQ	REPORT QUEUE (REPORTED MSS GO THERE)
183	/	TCB	TASK CONTROLBLOCK
184	/	REQ	REQUEST
185	/	RTN	RETURN
186	/	SCED	SCHEDULE
187	/	SCD	SCHEDULE
188	/	INH	INHIBITED
189	/	STL	STATIC TASK LIST
190	/	GPR	GENERAL PAGE 0 ROUTINES (SYSTEM PAGE 0),
191	/	STSK#	STATIC TASK NUMBER
192	/	PRIO	PRIORITY

```

193
194
195      /**** PAGE 0 ****
196      /
197      /PAGE 0 OF EACH VFLD IN WHICH NONPROTECTED TASKS ARE RUNNING
198      /IS DIVIDED IN TWO PARTS:
199      /      THE COMMON PART (LOC 14/147)
200      /      AND THE ONE TASK ONLY (=OTO) PART (LOC0-13, LOC150-177).
201      /THE COMMON PART CONTAINS:
202      /A.      GENERAL CONSTANTS
203      /B.      AN ENTRYPOINT FOR THE MONITOR (USED IN MONITOR CALLS)
204      /C.      THE FOLLOWING ROUTINES
205      /          VERHLT (EMERGENCY ERROR. STOP OPERATION AND TAKE
206      /                  APPROPRIATE ACTION)
207      /          VVCDF (DO VIRTUAL CHANGE DATA FIELD)
208      /          VVRDF (VIRTUAL READ DATAFIELD)
209      /          VVRCDF (DO CHANGE DATAFIELD TO REQUESTED VFLD)
210      /          VCDIF (CHANGE DATAFIELD TO THIS VFLD)
211      /          VMSNOD (REQUEST OR FREE A MESSAGE)
212      /          VGET (FETCH THE CONTENTS OF THE LOC POINTED AT
213      /                BY THE ARGUMENT, USING BASE AS OFFSET;
214      /                X POINTS TO THAT LOC AFTERWARDS)
215      /          VPUT (STORE THE CONTENTS OF AC INTO THE LOC
216      /                POINTED AT BY THE ARGUMENT, USING BASE AS
217      /                OFFSET;
218      /                X POINTS TO THE LOC AFTERWARDS)
219      /          VJUMS (CALL SUBROUTINE WHOSE NAME IS IN AC)
220      /                STORE SBRLINK IN LOC POINTED AT BY THE
221      /                FIRST LOC OF THE SBR, USING BASE AS
222      /                OFFSET;
223      /                X POINTS TO THAT LOC AFTERWARDS).
224      /D.      THE GENERAL PAGE 0 REGISTERS BASE AND X AND
225      /          PERHAPS MQ.
226      /
227      /THE OTO PART IS USED BY ONLY ONE TASK AT A TIME.
228      /A TASK CAN GET CONTROL OF THE OTO PART BY SETTING UP THE "ZERO-
229      /REQUESTBIT" IN STL[TSK, 1] BEFORE IT IS SWAPPED IN (USUALLY AT
230      /ASSEMBLY TIME). IT IS NOT ALLOWED TO HAVE CODE IN THE LOWER LOCS
231      /OF THE OTO PART.
232      /
233      /THE OTC PART OF VFLD 0 IS USED BY THE MONITOR.
234      /
235      /HERE FOLLOWS THE CODE OF VFLD 0 PAGE 0.

```

```

236          0000 FIELD 0
237          0000 *0 /INTERRUPT GOES HERE.
238 00000 0000 INTPC, 0 /!SAVE PC DURING INTERRUPT.
239          IFDEF PDP8E <
240 00001 6000 X6000, SKON /! ?
241 00002 5177 JMP INTR /!
242          7002 BSWR=BSW
243          >
244          IFNDEF PDP8E <
245          JMP INTR /!
246          BSWR=JMS I .
247          SBYTER
248          >
249 00003 7402 HLTINS, HLT /SOFT WARE ERROR!!!
250          /SYSHLT=JMS HLTINS-1
251
252 00004 0000 MONAC, 0 /AC DURING MONITOR CALL.
253 00005 0000 INTAC, 0 /SAVE AC DURING INTR
254 00006 0000 INTFL, 0 /INTFLAGS
255 00007 0000 MONFL, 0 /FLAGS DURING MONITOR CALL.
256          IFNZRO MONFL-MONAC-3 <ASS ERROR!!! MONAC-MONFL SHOULD BE 3>
257
258
259          0010 *10 /AUTO INDEXES.
260          /INITIALISATION CODE HERE!
261 00010 6213 X0, CIF CDF 10 /USED BY MONITOR.
262 00011 5412 X1, JMP I .+1 /USED BY MONITOR
263 00012 0177 X2, INIT-1 /!USED BY DEAF ROUTINES.
264          0200 INIT=200 /ENTRYPOINT OF INITIALISATION IN FLD 1.
265 00013 0000 X3, 0 /!USED BY DEAF ROUTINES.
266

```

```

267          /COMMON PART.
268          /          COPIED TO EACH VFLD IN WHICH NONPROTECTED TASKS ARE
269          /RUNNING. THE PRESENCE OF THIS PART IN A VFLD IS INDICATED BY THE
270          /MONINBIT IN THE FLDTAB.
271          /
272          /
273          0014  *14
274          0000  CUR=0          /CURRENT FIELD. MUST BE ADAPTED WHEN COPIED TO
275                               /OTHER FLDS.
276
277          /REQUEST OR FREE A MESSAGE.
278          /NOTE: DISTURBS LINK, DF:=IF!!!
279          /REQUEST:
280          /          CLA
281          /          MSREQ  /AC PTS AT MS[2] AT RETURN.
282          /FREE:
283          /          TAD    MSPTR  /AC PTS AT MS[2] (NEVER ZERO!)
284          /          MSFREE /AC = CONTENTS OF MS[2] AT RETURN.
285          4014  MSREQ=JMS .
286          4014  MSFREE=JMS .
287          00014 0000  VMSNOD, 0
288          00015 6002  IOF          /!ENTER STATE C,
289          00016 6203  XCDIF, CIF CDF 0          /!
290          00017 5514  JMP I    XMSNOD1 /!RETURNS IN TASK.
291
292
293          /MINIMUM TIME CDF ROUTINE FOR CDF TO REQUESTED VFLD.
294          /CALLING SEQUENCE:
295          /          VRCDF
296          /IF THE REQUESTED VFLD IS IN CORE WE JUST DO
297          /THE CDF, OTHERWISE CONTROL IS TRANSFERRED TO THE MONITOR,
298          /IN THE LATTER CASE THE TSK WILL EVENTUALLY BE RESCHEDULED
299          /WITH THE CORRECT DF.
300          /
301          4020  VRCDF= JMS .
302          /6 (10) CYCLES.
303          00020 0000  VVRCDF, 0          /TEMP FOR VVRCDF.
304          00021 6202  MYCIF, CIF CUR          /!ENTERING STATE C.
305          00022 4054  JMS          VCALL          /!OVERWRITTEN BY CDF.
306                               /MONITOR WILL RECOGNISE THIS CALL.
307          /          JMP I    VVRCDF
308
309          /HALT ON ERROR ROUTINE.
310          /CALLING SEQUENCE:
311          /          ERHLT  /STOP OPERATION; EMERGENCY ERROR!
312          /
313          4023  ERHLT= JMS .
314          00023 5420  VERHLT, JMP I    VVRCDF /MUST BE RESTORED AFTER EACH CALL!!
315          00024 4054  JMS          VCALL /-MON WILL RECOGNISE THIS SPECIAL CALL.
316
317          /VIRTUAL CHANGE DATAFIELD ROUTINE.
318          /CALLING SEQUENCE:
319          /          VCDF
320          /          NN          /6-BIT VFLD#.
321          /

```

```

322          /IF THE DESIRED VFLD IS ACTUALLY IN CORE THE CDF IS EXECUTED
323          /IMMEDIATELY, OTHERWISE CONTROL IS TRANSFERRED TO THE MONITOR AND
324          /THE TSK WILL BE RESCHEDULED EVENTUALLY HAVING THE CORRECT
325          /DATAFIELD.
326          4025  VCDF= JMS .
327          /28 (34) CYCLES.
328          /
329          00025 0000  VVCDF, 0
330          00026 6203          CIF CDF CUR      /!ENTER STATE C.
331          00027 3020          DCA      VVRCDF  /!SAVE AC FOR A WHILE.
332          00030 1425          TAD I      VVCDF  /!GET VFLD#
333          00031 2025          ISZ      VVCDF  /!
334          00032 1112          TAD      XFLDTAB /!
335          00033 3042          DCA      VCDFTM  /!
336          00034 6201  CCDF,  CDF 0          /!
337          00035 1442          TAD I      VCDFTM /!
338          00036 7500          SMA                      /!VFLD IN CORE NOW?
339          00037 4054  XVCALL, JMS      VCALL  /-MON WILL RECOGNISE THIS SPECIAL CALL.
340          00040 0117          AND      XCDF70 /!YES. GET FLD.
341          00041 3042          DCA      .+1      /!
342          00042 7402  VCDFTM, HLT                      /!EXECUTE THE CDF.
343          00043 1020          TAD      VVRCDF  /!
344          00044 5425          JMP I      VVCDF
345
346          /VIRTUAL READ DATAFIELD ROUTINE,
347          /CALLING SEQUENCE:
348          /          CLA
349          /          VRDF
350          /          ...          /RETURNS WITH 6-BIT VFLD# IN AC,
351          /DF:=IF!!!!
352          4045  VRDF= JMS .
353          00045 0000  VVRDF, 0
354          00046 6002          IOF                      /!ENTER STATE C.
355          00047 6202          CIF 0          /!
356          00050 5513          JMP I      XVRDF1 /!WILL RETURN IN TASK.
357
358          /NOTE: IN FLD 0 THIS ROUTINE IS USED TO SET THE DATAFLD TO THE
359          /USER'S IF.
360          0051  CDFUF=,
361          /CHANGE DATAFIELD TO INSTRUCTIONFIELD.
362          4051  CDFCUR= JMS .
363          00051 0000  VCDIF, 0
364          00052 6203  MYCDIF, CIF CDF CUR      /!ENTER STATE C.
365          00053 5451          JMP I      VCDIF
366
367          /MONITOR ENTRYPOINT,
368          /
369          /CALLING SEQUENCE:
370          /          TAD      ACARG      /GET ARG IN AC IF ANY.
371          /          CALL
372          /          FUNCTION/FUNCTION SPECIFICATION.
373          /          ARGS      /ARGUMENTS, IF ANY.
374          /          JMP      ERROR      /POSSIBLE ERRORRETURN.
375          /          ...          /NORMAL RETURN, IF ANY.
376

```

/APP. A1. MC8.1. PAGE 0, INTR SECTION PAL8-V9H 02/14/77 PAGE 7-2

377		4054	CALL=	JMS .	
378			MONPC,		/PC OF CALLING PROGRAM.
379	00054	0000	VCALL,	0	
380	00055	6202		CIF 0	/-
381	00056	6002		IOF	/-
382	00057	5511		JMP I XMONIT	/-
383					

```

384          /COMMON PART.
385          /REENTRANT TASK SUPPORT.
386
387          /GET ROUTINE.
388          /CALLING SEQUENCE:
389          /      GET
390          /          PTR          /VIRTUAL PTR.
391          /          /AC :=CONTENTS OF POINTED LOC,
392          /          X POINTS TO LOC AT RETURN.
393          /USES BASE AND X.
394
395          4060 GET= JMS .
396          /17 (19) CYCLES.
397          00060 0000 VGET, 0
398          00061 6203          CIF CDF CUR          /!ENTER STATE C.
399          00062 1107          TAD      BASE          /!
400          00063 1460          TAD I    VGET          /!
401          00064 3110          DCA      X              /!
402          00065 1510          TAD I    X              /!
403          00066 2060          ISZ      VGET          /!
404          00067 5460          JMP I    VGET
405
406          /SUBROUTINE CALLING ROUTINE.
407          /CALLING SEQUENCE:
408          /      CLA          /ARG GOES IN MQ
409          /      TAD      SUBNAME /AC PTS TO ROUTINE.
410          /      JUMS          /
411          /NOTE! THE FIRST LOC OF THE ROUTINE MUST CONTAIN THE ENTRY
412          /      OF THE REENTRANCY ARRAY WHERE THE SBRLINK MUST BE STORED.
413          /      USES BASE AND CHANGES X (POINTS TO LOC AFTERWARDS).
414          4070 JUMS= JMS .
415          /25 (29) CYCLES.
416          00070 0000 VJUMS, 0          /LUCKY. THIS IS A TEMP FOR VPUT!!!
417          00071 6203          CIF CDF CUR          /!ENTER STATE C.
418          00072 3075          DCA      VPUT          /!AS IF VPUT WERE CALLED TO STORE THE
419          00073 1070          TAD      VJUMS          /!SBRLINK, PREPARE ENTERING VPUT.
420          00074 7410          SKP          /!JUMP INTO VPUT. IT WILL DO A LUCKY
421          /!RETURN.
422
423          /PUT ROUTINE.
424          /CALLING SEQUENCE:
425          /      TAD      VAL          /GET VALUE TO BE STORED.
426          /      PUT
427          /          PTR          /VIRTUAL POINTER.
428          /          /STORE VALUE IN THE POINTED LOC,
429          /          X POINTS TO THE LOC AT RETURN.
430
431          4075 PUT= JMS .
432          /21 (23) CYCLES.
433          /JUMS ROUTINE JUMPS INTO IT!!!!!!
434          00075 0000 VPUT, 0
435          00076 6203          CIF CDF CUR          /!ENTER STATE C.
436          00077 3070          DCA      VJUMS          /!TEMP. MUST BE THAT ONE!!!
437          00100 1475          TAD I    VPUT          /!
438          00101 1107          TAD      BASE          /!

```

```

439 00102 3110      DCA      X      /!
440 00103 1070      TAD      VJUMS  /!
441 00104 3510      DCA      X      /!
442 00105 2075      ISZ      VPUT    /!
443 00106 5475      JMP      VPUT
444
445
446                IFZERO VPUT+1-100^4000 <ASS ERR!>
447
448                /PAGE 0 REGISTERS.
449 00107 0000      BASE,    0
450 00110 0000      X,      0
451                IFNDEF HARDMQ < MQ,      0      /IN CASE THERE IS NO EAE>
452
453                /SOME PTRS.
454 00111 3000      XMONIT, MONITOR
455 00112 4600      XFLDTAB,FLDTAB
456 00113 2671      XVRDF1, VRDF1
457 00114 2717      XMSNOD1,MSNOD1
458                4515      IEXIT=JMS I .
459 00115 0402      RPINTR
460 00116 4400      ZSTL,    STLST      /PTR FOR CONNECTED ROUTINES.
461                                /POINTER TO STL.
462 00117 6271      XCDF70, CDF 70      /USED TO RETURN ENTRIES IN STL.

```

463			/COMMON PART.	
464			/GENERAL CONSTANTS.	
465				
466			M7776,	
467	00120	0002	C2,	2
468			M7775,	
469	00121	0003	C3,	3
470			M7774,	
471	00122	0004	C4,	4
472			M7771,	
473	00123	0007	C7,	7
474	00124	0010	C10,	10
475	00125	0037	C37,	37
476	00126	0040	C40,	40
477	00127	0070	C70,	70
478	00130	0077	C77,	77
479	00131	0100	C100,	100
480	00132	0177	C177,	177
481	00133	0200	C200,	200
482	00134	0377	C377,	377
483				
484			M1,	
485	00135	7777	C7777,	7777
486			M2,	
487	00136	7776	C7776,	7776
488			M3,	
489	00137	7775	C7775,	7775
490			M4,	
491	00140	7774	C7774,	7774
492			M5,	
493	00141	7773	C7773,	7773
494			M6,	
495	00142	7772	C7772,	7772
496			M10,	
497	00143	7770	C7770,	7770
498			M20,	
499	00144	7760	C7760,	7760
500			M100,	
501	00145	7700	C7700,	7700
502			M200,	
503	00146	7600	C7600,	7600
504				
505			IFNZRO 150-..4000 <ASS ERR!>	

```

506          /ONE TASK ONLY PART.
507
508          /FLD 0: MONITOR ROUTINES AND CONSTANTS.
509          0150  *150
510
511
512          /4 TEMPORARIES FOR MONITOR.
513          00150 0000 MON0, 0
514          00151 0000 MON1, 0
515          00152 0000 MON2, 0
516          00153 0000 MON3, 0
517
518          /TEMPORARIES FOR INTERRUPT SECTION AND DEAF ROUTINES.
519          00154 0000 INT0, 0      /!
520          00155 0000 INT1, 0      /!
521          00156 0000 INT2, 0      /!
522          00157 0000 INT3, 0      /!
523
524          /MONITOR CALL PARAMETERS.
525          00160 0000 MONFUNC,0      /MONITOR FUNCTION WORD
526          00161 0000 MARG1, 0      /MONITOR ARG1
527          00162 0000 MARG2, 0      /MONITOR ARG2.
528
529          00163 0000 VCURIF, 0      /CURRENT VINSTRFLD IN B0-5.
530          00164 0000 CURTSK, 0      /PTR TO TCB[5] OF CURRENT TSK.
531          00165 4700 HPRI0, R&PTRS /HIGHEST RUNNABLE PRIORITY
532          00166 0000 SCDREQ, 0      /SCHEDULE REQUEST FLAG.
533                                     /0: SCDREQ PENDING, OTHER: NO SCDREQ PENDING,
534          00167 0000 SCDINH, 0      /SCHEDULE INHIBIT FLAG.
535                                     /0: SCHEDULING ALLOWED
536                                     />=1: SCHEDULING INHIBITED.
537          /NOTE: WHEN SCDINH IS CLEARED SCDREQ MUST ALWAYS
538          /BE CHECKED!
539
540          00170 4722 ZCORMAP,CORMAP
541          00171 1000 ZSNDREP,SNDREP      /PTR TO SNDREP ROUTINE.
542          00172 4305 ZFN, FN      /PTR TO FN ROUTINE.
543          00173 1000 Z1000, 1000
544
545          0000 AVPT=0      /INITIALISE AVAILLIST BUILDER.
546
547          IFNZRO 176-..4000 <ASS ERR!>
548
549          /***** END OF PAGE 0 *****/
550          0176  *176
551          4576  SYSHLT=JMS I
552          00176 0002      HLTINS-1

```

```

553          /***** INTERRUPT SECTION *****/
554
555          0177 *177
556          /SAVE STATUS.
557          IFDEF PDP8E <
558          00177 3005 INTR, DCA INTAC /!
559          / GTF /!
560          6004 INS200=GTF /THIS INSTRUCTION WILL BE STORED HERE BY
561          /INIT.
562          00200 5010 JMP X0 /TO START UP INIT. GETS OVERWRITTEN,
563          /ASSERTION: BIT2=1, BIT3-4=0. WE DONT USE BIT2.
564          / AND C6177 /! REMOVE DIRTY FLAGS,
565          00201 3006 DCA INTFL /!
566          >
567          IFDEF PDP8I <
568          INTR, DCA INTAC /!
569          / RAR /!
570          INS200=RAR
571          JMP X0 /!TO START UP INIT. GETS OVERWRITTEN,
572          RIB /!READ SAVE FIELD REGISTER,
573          DCA INTFL /!
574          >
575          INTR2, /!COME HERE IF ANOTHER FLAG IS PENDING,
576
577          IFDEF STATX <
578          00202 2301 ISZ INTCT /!
579          00203 5207 JMP SKPCHN /!
580          00204 2302 ISZ INTCT+1 /!
581          00205 5207 JMP SKPCHN /!
582          00206 2303 ISZ INTCT+2 /!OVERFLOW EACH 2**14 HOURS,
583          / JMP SKPCHN /!
584          >
585
586          /ALLOCATE THE SKIPCHAIN.
587          IFNDEF SKPCHN <
588          SKPCHN=, /IF SKIPCHAIN FITS ON CURRENT PAGE.
589
590          SKPEND=MAXDEV+5+.+2 /END OF SKIPCHAIN.
591          IFDEF CHECK <SKPEND=SKPEND+1> /1 LOC FOR EXTRA CHECK.
592          SKPFIT=400 /SKIPCHAIN MUST FIT BEFORE 400.
593          IFDEF STATX <SKPFIT=SKPFIT-3> /RESERVE 3 LOCS FOR COUNTER.
594          NONFIT= SKPFIT-SKPEND^4000
595
596          IFNZRO NONFIT < /IF SKIPCHAIN EXCEEDS CURRENT PAGE
597          IFDEF STATX <SKPCHN=SKPCHN+4 > /A JMP AND 3 COUNTERS,
598
599          /PROPER ALLOCATION FOR PAGE CROSSING.
600          /FORCE AN INTSLOT AT LOC 377
601          QUOT5=377-SKPCHN/5+5 /5 LOCS FOR EACH INTSLOT.
602          SKPSHF=377-QUOT5-SKPCHN
603          SKPCHN=SKPCHN+SKPSHF
604          >
605          >
606
607          IFNZRO NONFIT <

```

```

608      IFNZRO SKPCHN-, <JMP SKPCHN>
609      IFDEF STATX <INTCT, 0;0;0>
610      >
611
612
613
614
615      /***** SKIPCHAIN *****/
616      /HERE COMES THE SKIPCHAIN. IT IS A LIST OF
617      /SOCALLED INTSLOTS. AN INTSLOT CONSISTS OF 5 CONSECUTIVE LOCS,
618      /ON THIS PAGE THERE IS ROOM FOR ABOUT 22 INTSLOTS, IF ONE NEEDS
619      /MORE, THE SKIPCHAIN MUST BE CAREFULLY ALLOCATED TO
620      /CROSS THE PAGEBOUND PROPERLY.
621      /THE ORDER OF THE SLOTS IN THE SKPCHN IS DETERMINED
622      /IN THE CONFIGURATIONSECTION AT THE LABEL INTDEF.
623      /
624      /THE FIRST TWO LOCS OF AN INTSLOT CONTAIN:
625      /      SKIPIOT; JMP .+4
626      /THE CONTENTS OF THE LAST THREE LOCS IS CONTROLLED BY THE
627      /FOLLOWING MONITOR FUNCS:
628      /CLINTR (CLAIM INTSLOT).
629      /      CLEARIOT
630      /      IEXIT
631      /      MSPTR
632      /      A TSK CAN GET CONTROL OF AN INTSLOT BY EXECUTING THIS
633      /      MONFUNC. A MS IS RESERVED WHOSE PTR IS DENOTED IN THE
634      /      INTSLOT. EACH TIME AN INT OCCURS THIS MS IS REPORTED TO
635      /      THE TSK. NO TWO TSKS MAY LOOK AT THE SAME INTSLOT,
636      /FRINTR (FREE INTSLOT).
637      /      CLEARIOT
638      /      IEXIT
639      /      0
640      /      SET INTSLOT IN DISCARD MODE. CAN BE USED AFTER A CONNECT,
641      /      CLAIM OR DISABLE FUNCTION. IF THE CORRESPONDING DEVICE
642      /      INTERRUPTS THE CLEARIOT IS EXECUTED, AC IS CLEARED AND
643      /      THE FAST EXIT IS TAKEN.
644      /DSINTR (DISABLE INTSLOT).
645      /      JMP .+3
646      /      0
647      /      0
648      /      IF A TSK DISABLES THE INTERRUPTFUNCTION OF SOME DEVICE,
649      /      IT MUST DISABLE THE INTSLOT AS WELL. IF NOT, GREAT
650      /      PROBLEMS WILL ARISE IN TREATING DEVICES CORRECTLY. WHEN
651      /      THE INTSLOT IS DISABLED FLAGS OF THAT DEVICE WILL BE
652      /      IGNORED. NOTE: IF A ROUTINE IS CONNECTED TO THE INTSLOT,
653      /      THAT ROUTINE MAY IGNORE THE DEVICEFLAG BY TAKING A SIMPLE
654      /      RETURN. IN THAT CASE SEARCHING THE SKIPCHAIN IS
655      /      CONTINUED.
656      /CNINTR (CONNECT TO INTSLOT).
657      /      CIF CDF FLD
658      /      JMS I .+1
659      /      SBRNAME
660      /      CONNECT A DEAF ROUTINE TO THE INTSLOT.
661      /      EACH TIME THE DEVICEFLAG IS ENCOUNTERED IN THE SKPCHN,
662      /      THAT ROUTINE IS CALLED. IF THE DEVICE INTERRUPT FUNCTION

```

/APP. A1. MC8.1. PAGE 0, INTR SECTION PAL8-V9H 02/14/77 PAGE 11-2

```
663      /      WAS DISABLED, IT MUST IMMEDIATELY RETURN FROM THAT
664      /      ROUTINE, THUS ENABLING CORRECT TREATMENT OF OTHER
665      /      DEVICES, (THE SBRLINK MUST BE INCREMENTED ONCE BEFORE
666      /      RETURNING!!!).
667      /      NORMALLY A CONNECTED ROUTINE IS TERMINATED BY EXECUTING
668      /      THE IEXIT INSTRUCTION, SPECIFYING A MS TO BE REPORTED
669      /      IN THE NEXT LOC. IF NO MS NEED BE REPORTED, 0000 IS
670      /      SPECIFIED.
671      /      NOTE: CONNECTED ROUTINES MUST ALWAYS BE IN CORE, IT IS
672      /      NOT ALLOWED TO TOUCH THE ION IN A CONNECTED ROUTINE.
673      /
674      /MOST INTSLOTS ARE INITIALLY EITHER FREE OR DISABLED, DEPENDING
675      /ON THE DEFAULT ATTITUDE OF THE DEVICE.
676      /SOME ARE ALREADY CLAIMED BY OR CONNECTED TO MONTSKS.
```

```

677      IFDEF TT1 <
678      0240      *TT1+5+SKPCHN
679      00240    6041      TSF      /!
680      00241    5245      JMP      .+4    /!
681      00242    6042      TCF      /!
682      00243    4515      IEXIT      /!
683      00244    0000      0          /!
684      >
685
686      IFDEF KB1 <
687      0245      *KB1+5+SKPCHN
688      00245    6031      KSF      /!
689      00246    5252      JMP      .+4    /!
690      00247    6036      KRB      /!PERHAPS A PITY FOR THE PDP8/I
691      00250    4515      IEXIT      /!
692      00251    0000      0          /!
693      >
694
695      IFDEF TT2 <
696      0252      *TT2+5+SKPCHN
697      00252    6431      TSF2     /!
698      00253    5257      JMP      .+4    /!
699      00254    6432      TCF2     /!
700      00255    4515      IEXIT     /!
701      00256    0000      0          /!
702      >
703
704      IFDEF KB2 <
705      0257      *KB2+5+SKPCHN
706      00257    6421      KSF2     /!
707      00260    5264      JMP      .+4    /!
708      00261    6426      KRB2     /!
709      00262    4515      IEXIT     /!
710      00263    0000      0          /!
711      >
712
713      IFDEF TT3 <
714      0264      *TT3+5+SKPCHN
715      00264    6471      TSF3     /!
716      00265    5271      JMP      .+4    /!
717      00266    6472      TCF3     /!
718      00267    4515      IEXIT     /!
719      00270    0000      0          /!
720      >
721
722      IFDEF KB3 <
723      0271      *KB3+5+SKPCHN
724      00271    6461      KSF3     /!
725      00272    5276      JMP      .+4    /!
726      00273    6466      KRB3     /!
727      00274    4515      IEXIT     /!
728      00275    0000      0          /!
729      >
730
731      IFDEF DTA <

```

```

732      *DTA↑5+SKPCHN
733          DTSF          /!
734          JMP          .+4    /!
735          JMP          .+3    /!
736          0            /!
737          0            /!
738      >
739
740      IFDEF ANALEX <
741      *ANALEX↑5+SKPCHN
742          PRSF          /!
743          JMP          .+4    /!
744          JMP          .+3    /!
745          0            /!
746          0            /!
747      >
748
749      IFDEF RK8E <
750      *RK8E↑5+SKPCHN
751      00207 0207      DSKP          /!
752      00210 5214      JMP          .+4    /!
753          IFNZRO SYDISK-RK8E <
754          JMP          .+3    /!DISABLED INITIALLY.
755          0            /!
756          0            /!
757      >
758      IFZERO SYDISK-RK8E <
759      00211 6203      CIF CDF 0      /!CONNECTED TO SYWAIT ROUTINE.
760      00212 4613      JMS I          .+1 /!CALL CONNECTED ROUTINE
761      00213 2333      RKINTR /!
762      >
763      >
764
765      IFDEF RF08 <
766      *RF08↑5+SKPCHN
767          6623          /!
768          JMP          .+4    /!
769      IFNZRO SYDISK-RF08 <
770          JMP          .+3    /!DISABLED INITIALLY.
771          0            /!
772          0            /!
773      >
774      IFZERO SYDISK-RF08 <
775          CIF CDF 0      /!CONNECTED ROUTINE.
776          JMS I          .+1 /!
777          RFINTR        /!
778      >
779      >
780
781      IFDEF DK8EP <
782      00233 0233      *DK8EP↑5+SKPCHN
783          CLSK          /!
784      00234 5240      JMP          .+4    /!
785          IFNZRO SYTIME-DK8EP <
786          JMP          .+3    /!DISABLED INITIALLY.

```

```

787          0          /!
788          0          /!
789      >
790      IFZERO SYTIME-DK8EP <
791      00235 6135      CLSA          /! CLAIMED BY MTTIME.
792      00236 4515      IEXIT        /!
793      00237 2666          TIMEMS    /!
794      >
795      >
796
797      IFDEF MULTIP < /PDP-8/I MULTIPLEXER.
798      *MULTIP+5+SKPCHN
799          T1SKP        /!
800          JMP          .+4        /!
801      IFNZRO SYTIME-MULTIP <
802          T1ON          /! A PITY!
803          IEXIT        /!
804          0            /!
805      >
806      IFZERO SYTIME-MULTIP <
807          CIF CDF 0      /!
808          JMS          .+1        /!
809          MULINTR        /!
810      >
811      >
812
813      IFDEF PDP8IE < /MC PDP8/I-PDP8/E INTERFACE,
814      0221      *PDP8IE+5+SKPCHN      /THESE TWO DEVICES GO TOGETHER,
815      00221 6571      P8SIF          /! INPUT?
816      00222 5226      JMP          .+4        /!
817      00223 5226      JMP          .+3        /! DISABLED INITIALLY.
818      00224 0000      0            /!
819      00225 0000      0            /!
820      00226 6573      P8SOF          /! OUTPUT?
821      00227 5233      JMP          .+4        /!
822      00230 5233      JMP          .+3        /! DISABLED INITIALLY.
823      00231 0000      0            /!
824      00232 0000      0            /!
825      >
826
827      IFDEF PDP11 < /MC PDP-8 PDP-11 INTERFACE,
828      *PDP11+5+SKPCHN      /TWO DEVICES!
829          P11SOF        /! OUTPUT?
830          JMP          .+4        /!
831          JMP          .+3        /! DISABLED INITIALLY
832          0            /!
833          0            /!
834          P11SIF        /! INPUT?
835          JMP          .+4        /!
836          JMP          .+3        /! DISABLED INITIALLY.
837          0            /!
838          0            /!
839      >
840
841      IFDEF FPTP <

```

```

842          *FPTP+5+SKPCHN
843          FPSF          /!
844          JMP          .+4      /!
845          FPCR          /!
846          IEXIT        /!
847          0            /!
848      >
849
850      IFDEF FPTR <
851      *FPTR+5+SKPCHN
852          FRSF          /!
853          JMP          .+4      /!
854          FPCR          /!
855          IEXIT        /!
856          0            /!
857      >
858
859      IFDEF KM8E <
860      *KM8E+5+SKPCHN
861      00214 6254      SINT          /!USER MODE.
862      00215 5221      JMP          .+4      /!
863      00216 6204      CINT          /!TYPICAL RESULT IF NOT CONNECTED.
864      00217 4515      IEXIT        /!
865      00220 0000      0            /!
866      >
867
868      0276 *MAXDEV+5+SKPCHN
869      00276 0000      0            /!SIGNAL END OF SKPCHN; INNOCENT.
870
871      00277 6004      IFDEF CHECK <
872      IFDEF PDP8E <GTF> /SHOW STATUS IN AC,
873      IFDEF PDP8I <RIS> /SHOW STATUS IN AC, (MC ONLY;)
874      00300 4576      >          SYSHLT          /UNKNOWN INTERRUPT.
875
876      IFDEF STATX <
877      00301 0000      IFZERO NONFIT <INTCT, 0;0;0>
878      00302 0000
879      00303 0000
880      >
881
882      IFNZRO .-1^177-175^4000 < /AT LEAST 3 FREE LOCS ON THIS PAGE
883      00304 0000      AVPT;(0;AVPT=.-2
884      00305 0377
885      0304
886      >
887
888      00377 0000
889      0400      PAGE
890

```

```

891 00400 0000 IFDEF STATX <INTSCD, 0;0>
892 00401 0000
893
894 /IEXIT ROUTINE. LEAVE INTERRUPT SECTION.
895 /CALLED EITHER DIRECTLY FROM SKPCHN, OR FROM CONNECTED ROUTINES.
896 /ARG1 =MSPTR, PTR TO MS[2].
897 /STORE AC (POSSIBLE I/O STATUS) IN MS[2].
898 /IF MSPTR=0 NO MS NEED BE REPORTED.
899 /CALLED WITH DF TO CALLING FIELD.
900 00402 0000 RPINTR, 0 /!POSSIBLE I/O STATUS IN AC.
901 00403 3155 DCA INT1 /!
902 00404 1602 TAD I RPINTR /!
903 00405 6201 CDF 0 /!CONNECTED ROUTINES MIGHT CALL US FROM
904 /!OTHER FIELDS.
905 00406 7450 SNA /!IF NO MS ATTACHED, THE SLOT IS FREE
906 00407 5217 JMP FSTEXT /!AND WE JUMP DIRECTLY TO THE END.
907 00410 3154 DCA INT0 /!PTR TO MS[2]
908 00411 1155 TAD INT1 /!STATUS TO MS
909 00412 3554 DCA I INT0 /!
910 00413 7344 ACM2 /!
911 00414 1154 TAD INT0 /!
912 00415 3154 DCA INT0 /!PTR TO MS[0] SENDER WORD.
913 /!CANNOT BE 0 FOR INTERRUPT MS.
914 00416 4571 JMS I ZSNDREP /!REPORT.
915
916
917 FSTEXT,
918 IFDEF PDP8E <
919 00417 6003 SRQ /!SAVING TAKES ABOUT 60 CYCLES. THIS
920 00420 7410 SKP /!CHECK DECREASES RESPONSE TIME.
921 >
922 IFDEF PDP8I <
923 RIS /!READ INTERRUPT STATUS. ONLY MC!!
924 SPA CLA /!
925 >
926 00421 5777' JMP INTR2 /!
927
928 5776' EXTPRV=JMP MONEX
929 00422 0000 EXTALL, 0 /!EXIT ALLOWED SWITCH.
930 /CHANGED INTO JMP MONEX BY SIMINTR
931 /!COMMAND.
932 00423 1166 TAD SCDREQ /!SCDREQ PENDING
933 00424 1167 TAD SCDINH /!AND SCHEDULING NOT INHIBITED?
934 00425 7640 SZA CLA /!
935 00426 5245 JMP FSTXT2 /!NO. LEAVE INTR SECTION.
936 /YES. CHECK HOW TO SAVE.
937
938 00427 1164 TAD CURTSK /!IF NO TASK RUNS WE SCHEDULE WITHOUT
939 00430 7650 SNA CLA /!SAVING
940 00431 5775' JMP SCED /!
941 IFDEF STATX < /INTSCD IS ONLY COUNTED IF SOME TSK IS
942 00432 2200 ISZ INTSCD /!//INTERRUPTED.
943 00433 5236 JMP .+3 /!
944 00434 2201 ISZ INTSCD+1 /!
945 00435 7000 NOP /!

```

```

946      >
947      00436 4051      JMS      CDFUF      /*IF USERMODE SKP, ELSE CDF INSTRFLD,
948      00437 7410      SKP          /*NO USERMODE.
949      00440 5264      JMP      ITSAF      /*NO TROUBLES, SAVE.
950      00441 1000      TAD      INTPC      /*MIND: DF NONZERO!!!
951      00442 1374      TAD      (-VCALL-1      /*DONT SWITCH TSK IF PC=VCALL+1
952      00443 7640      SZA CLA          /*FOR SAVING IS DIFFICULT AND SCDREQ WILL
953                                     /*BE TESTED SOON ANYHOW.
954      00444 5252      JMP      TSKSWT      /*NO; GO SWITCHING.
955
956      FSTXT2,
957      IFDEF PDP8E <
958      00445 1006      TAD      INTFL      /*MIND: DF NONZERO!!!!
959      00446 6005      RTF          /*
960      00447 7200      CLA          /*
961      00450 1005      TAD      INTAC      /*
962      00451 5400      JMP      INTPC      /*
963      >
964      IFDEF PDP8I <
965      TAD      INTFL      /*FETCH LINK.
966      RAL          /*
967      CLA          /*
968      TAD      INTAC      /*
969      RMF          /*
970      ION          /*
971      JMP      INTPC      /*
972      >
973
974
975
976      /TASK SWITCHING.
977      00452 1000      TSKSWT, TAD      INTPC      /*INTERRUPTED IN ONE OF THE GENERAL
978      00453 0145      AND      C7700      /*PAGE 0 ROUTINES?
979      00454 7640      SZA CLA          /*
980      00455 5264      JMP      ITSAF      /*GO FOR SIMPLE SAVE INTRSTATUS.
981
982      00456 7240      CLA CMA          /*INTERRUPTED IN GENERAL PAGE 0 ROUTINE.
983      00457 1000      TAD      INTPC      /*SAVE STATUS BEFORE CALL.
984      00460 3000      DCA      INTPC      /*
985      00461 7240      CLA CMA          /*MIND: DF STILL TO USERS' INSTRFLD!!!
986      00462 1400      TAD      INTPC      /*PC BEFORE CALL
987      00463 3000      DCA      INTPC      /*
988      /FALL INTO SAVE ROUTINES.
989

```

```

990      /***** SCHEDULING SCHEME AND SAVING. *****/
991      /
992      /THESE FOUR ARE THE IMPORTANT VARIABLES IN THE SCHEDULER:
993      /CURTSK POINTER TO TCB[5] (LINKWORD) OF THE TASK CURRENTLY
994      /RUNNING
995      /HPRIO INDICATING THE HIGHEST RUNNABLE PRIORITY
996      /SCDREQ SCHEDULE REQUESTFLAG, 0: SCEDREQ PENDING
997      /SCDINH SCHEDULING INHIBITION FLAG, >=1: SCHEDULING INHIBITED.
998      /
999      /CURTSK=0 IF NO TASK IS RUNNING (JUST AFTER INITIALIZATION,
1000     / WHILE SCHEDULING AND PERHAPS WHILE IDLING).
1001     /HPRIO INDICATES THE HIGHEST RUNNABLE PRIORITY BY POINTING IN ONE
1002     / OF THE PRIORITY RUNQS. IF NO TASKS ARE RUNNABLE IT POINTS
1003     / IN THE RUNQ CONTAINING THE IDLELOOP. IN GENERAL THE TASK
1004     / CURRENTLY RUNNING IS THE FIRST IN THE RUNQ POINTED AT BY
1005     / HPRIO.
1006
1007     /TASKS OF A HIGHER PRIORITY THAN THE CURRENT ONE CAN
1008     /BECOME RUNNABLE AS A CONSEQUENCE OF AN INTERRUPT OR AS A
1009     /CONSEQUENCE OF SOME MONITOR CALL (FOR INSTANCE A MESSAGE
1010     /REPORT). THEREFORE BEFORE LEAVING THE INTERRUPT SECTION OR
1011     /THE MONITOR WE CHECK THAT THE CURRENT TASK HAS STILL THE
1012     /HIGHEST PRIORITY OF ALL RUNNABLE TASKS. IF IT HAS NOT, THE
1013     /SYSTEM SWITCHES TASKS AS SOON AS POSSIBLE.
1014     /AFTER A MONITOR CALL THE CURRENT TASK MAY NO LONGER BE
1015     /RUNNABLE (FOR INSTANCE AFTER A SEND MESSAGE AND WAIT
1016     /REPORT). THE STATUS IS SAVED AND THE TASK IS REMOVED FROM
1017     /THE RUNQ. SUBSEQUENT THE SCHEDULER IS ENTERED,
1018     /
1019     /***** SAVING.
1020     /BEFORE A NEW TASK CAN BE RUN THE CURRENT TASKS' STATUS MUST BE
1021     /SAVED. THE STATUS IS SAVED AS FOLLOWS:
1022     /VIF+VDF      TCB[7]  VIRTUAL FLDS.
1023     /AC           TCB[8]  ACCUMULATOR.
1024     /PC           TCB[9]  PROGRAM COUNTER.
1025     /MQ           TCB[10] MULTIPLIER QUOTIENT.
1026     /EAE         TCB[11] EAE STATUS +LINK.
1027     /BASE        TCB[12] VALUE OF BASE (NOT IN PROTECT TASKS).
1028     /X           TCB[13] VALUE OF X (NOT IN PROTECT TASKS).
1029     /PROTECT TASKS ARE TASKS RUNNING WITH USER MODE ON.
1030     /AT MONITOR ENTRANCE PART OF THE STATUS IS SAVED IN
1031     /MONPC, MONAC AND MONFL (FLDS +L +SOME EAE). THIS IS USED IF
1032     /SAVING IS REQUIRED AFTER THIS CALL.
1033     /AT INTERRUPT PART OF THE STATUS IS SAVED IN
1034     /INTPC, INTAC AND INTFL (FLDS +L +SOME EAE). THIS IS USED
1035     /WHEN TASK SWITCHING OCCURS AFTER THE INTERRUPT.
1036     /
1037     /***** SCHEDULING.
1038     /IT SHOULD BE CLEAR THAT WE ARE NOT ALLOWED TO SWITCH TASKS
1039     /IF THE INTERRUPT CAME DURING A MONITOR CALL. TWO THINGS
1040     /MIGHT GO WRONG: MONITOR CALLS OF DIFFERENT TASKS CAN INTERFERE,
1041     /AND INTPC, INTAC AND INTFL DO NOT CONTAIN THE PROPER STATUS.
1042     /THIS IS REMEDIED BY THE VARIABLE SCDINH. WHEN WE ENTER A
1043     /CRITICAL SECTION (FOR INSTANCE AT MONITOR CALL) IN WHICH WE DONT
1044     /LIKE TO BE INTERRUPTED BY OTHER TSKS WE SET SCDINH TO 1, THEREBY

```

```

1045 /INHIBITING SCHEDULING. AT THE END OF THE CRITICAL SECTION WE
1046 /CLEAR SCDINH AND CHECK SCDREQ TO SEE WETHER A SCHEDULE REQUEST
1047 /IS STILL PENDING. IF IT IS WE SAVE THE CURRENT TSK AND ENTER
1048 /THE SCHEDULER.
1049 /TASKS ARE ENTERED IN THE RUNQ BY CALLING INRUNQ.
1050 /THIS ROUTINE CHECKS WETHER THE TSK INSERTED HAS A HIGHER PRIO
1051 /THAN THE CURRENT TSK. IF SO IT CLEARS SCDREQ, SIGNALLING THE
1052 /SCHEDULE REQUEST. AT THE END OF THE INTR SECTION AND WHEN
1053 /LEAVING THE MONITOR THIS VARIABLE IS TESTED.
1054 /
1055 /AFTER SAVING WE INSPECT THE PRIORITY RUNQS ONE BY ONE
1056 /STARTING AT THE ONE INDICATED BY HPRI0. IF NECESSARY
1057 /HPRI0 IS UPDATED. WHEN WE FIND A RUNNABLE TASK WE CHECK FOR ITS
1058 /FLDS TO BE IN CORE. IF THEY ARE THE TASK IS STARTED. OTHERWISE
1059 /WE ISSUE A SWAPREQ (SEE FCHECK AND MTSWAP)
1060 /AND CONTINUE SEARCHING THE RUNQS TO FIND THE NEXT IMPORTANT
1061 /TASK. IN THIS SITUATION WE MIGHT START A TASK THAT IS NOT THE
1062 /FIRST IN ITS RUNQ.
1063 /
1064 /NOTE!!!!: THREE PITFALLS:
1065 /1. WE DONT ISSUE A SWAP IF ANOTHER SWAPREQ WAS STILL PENDING.
1066 / IF WE DID THE SYSTEM MIGHT BE SWAPPING WITHOUT EVER RUNNING
1067 / A TASK.
1068 /2. THE INTERRUPT THAT INDICATES SWAP COMPLETION MAKES
1069 / THE MONITOR SWAP TASK RUNNABLE. THIS CAUSES HPRI0 TO POINT
1070 / AT THE PRIO 0 RUNQ, THUS GUARANTEEING THAT ALL RUNQS ARE
1071 / REINSPECTED. THE TASK THAT CAUSED THE SWAP REQUEST IS
1072 / PROBABLY THE FIRST TO BENEFIT FROM THIS.
1073 /3. IF THE TASK WHOSE FLDS ARE NOT IN CORE IS REENTRANT
1074 / (BIT1 OF TCB[6] SET), WE SKIP OVER ALL THE REST OF THAT
1075 / PRIO RUNQ. IF WE DIDNOT DIFFERENT VERSIONS OF THAT
1076 / REENTRANT TASK MIGHT INTERRUPT ONE ANOTHER AT INPREDICTABLE
1077 / MOMENTS.
1078
1079

```

```

1080      /SAVE AFTER INTERRUPT.
1081      00464 6201  ITSAF, CDF 0      /!MAKE SURE ABOUT DF.
1082      00465 1006      TAD      INTFL /!COPY INTR STATUS INTO MONTIORSTATUS.
1083      00466 3007      DCA      MONFL /!
1084      00467 1005      TAD      INTAC /!
1085      00470 3004      DCA      MONAC /!
1086      00471 1000      TAD      INTPC /!
1087      00472 3054      DCA      MONPC /!
1088      00473 5300      JMP      CMNSAF /!JUMP INTO COMMON SAFE.
1089
1090
1091      MONOUT,      /SAVE CURTSK AND SCHEDULE.
1092      IFDEF STATX <
1093      00474 2343      ISZ      MONSCD
1094      00475 5300      JMP      .+3
1095      00476 2344      ISZ      MONSCD+1
1096      00477 7000      NOP
1097      >
1098      CMNSAF,      /COMMON SAFE ROUTINE.
1099      MONSAF, TAD      CURTSK /!\\?SCDINH>=1, OR DEAF.
1100      00501 7001      IAC      /!\\?
1101      00502 3010      DCA      X0 /!\\?
1102      00503 1007      TAD      MONFL /!\\?SAFE STATUS AFTER MONCALL.
1103      00504 0123      AND      C7 /!\\?
1104      00505 1170      TAD      ZCORMAP /!\\?PTR TO ACTUAL FLD.
1105      00506 3150      DCA      MON0 /!\\?
1106      00507 1550      TAD      MON0 /!\\?GET VFLD#
1107      00510 0130      MONSF2, AND C77 /!\\?SAFE AFTER FCHECK JUMPS HERE.
1108      00511 1163      TAD      VCURIF /!\\?ADD VINSTRFLD
1109      00512 3410      DCA      X0 /!\\?
1110      00513 1004      TAD      MONAC /!\\?
1111      00514 3410      DCA      X0 /!\\?
1112      00515 1054      TAD      MONPC /!\\?
1113      00516 3410      DCA      X0 /!\\?
1114      IFDEF PDP8E <
1115      IFDEF EAE <
1116      00517 7441      SCA      /!\\?
1117      00520 7521      SWP      /!\\?SC TO MQ
1118      00521 3410      DCA      X0 /!\\?MQ
1119      00522 7403      SCL      /!\\?SKIPS IF MODE A
1120      00523 1133      TAD      C200 /!\\?
1121      00524 1007      TAD      MONFL /!\\?NOTE: 200 NOT IN FLAGS!,
1122      00525 0145      AND      C7700 /!\\?
1123      00526 7501      MGA      /!\\?ORS STEPCOUNT IN LOW ORDER BITS
1124      00527 3410      DCA      X0 /!\\?
1125      >
1126      >
1127      IFDEF PDP8I <
1128      IFNDEF EAE <
1129      JMS      CDFUF /!\\?SKIPS IF USER MODE.
1130      TAD      (MQ /!\\?
1131      CDF 0 /!\\?
1132      DCA      X0 /!\\?
1133      TAD      MONFL /!\\?
1134      AND      C7700 /!\\?

```

```

1135          DCA I    X0      /\?
1136          >
1137          >
1138      00530 4051      JMS      CDFUF  /\?IF USERMODE SKP, ELSE CDF INSTRFLD.
1139      00531 7410      SKP          /\?
1140      00532 5775'     JMP      SCED  /\?DONT TOUCH LOCS IN USERS IF,
1141      00533 1773      TAD I    (X      /\?
1142      00534 7421      STORMQ      /\?
1143      00535 1772      TAD I    (BASE  /\?
1144      00536 6201      CDF 0      /\?
1145      00537 3410      DCA I    X0      /\?
1146      00540 7701      GETMQ      /\?
1147      00541 3410      DCA I    X0      /\?
1148      00542 5775'     JMP      SCED  /\?JUMP TO THE SCHEDULER.
1149      00543 0000      IFDEF STATX <MONSCD, 0;0 >
1150      00544 0000
1151
1152          /SET CURRENT TSK TO WAIT; WAITCONDITIONS IN AC.
1153      00545 0000      CURWT, 0
1154      00546 6002          IOF          /\
1155      00547 3157          DCA      INT3  /\WAITCONDITIONS IN INT3.
1156      00550 7040          CMA          /\
1157      00551 1164          TAD      CURTSK /\PTR TO TCB[4] IN AC.
1158      00552 4771'        JMS      OURUNG /\
1159      00553 5745          JMP I    CURWT /\
1160
1161      00554 1051      TSKSAF, TAD      CDFUF  /\IF WE CAME JUST FROM THE SCDULER
1162      00555 7650          SNA CLA      /CDFUF IS STILL 0
1163      00556 5775'        JMP      SCED  /AND WE NEED NOT SAVE STATUS.
1164      00557 5300          JMP      MONSAF
1165
1166          IFNDEF PDP8E <
1167          SBYTER, 0
1168          CLL RTR;RTR;RTR
1169          JMP I    SBYTER
1170      >
1171
1172      00560 0304          AVPT;(0;AVPT=-2
1173      00561 0570
1174      0560
1175
1176      00570 0000
1177      00571 1136
1178      00572 0107
1179      00573 0110
1180      00574 7723
1181      00575 0617
1182      00576 0731
1183      00577 0202
1184          0600      PAGE

```

```

1185          /A TSK WAS SCHEDULED THAT IS ONDISK (TCB[6] B0 SET).
1186          /SET TSK TO WAIT WITH FWTSWP CONDITION AND SEND A MS TO
1187          /MTFTCH TO FETCH THE TSK FROM DISK.
1188 00600 1377 ONDISK, TAD      (FWTSWP
1189 00601 4776 JMS      CURWT  /\CALLING DEAF ROUTINES AND CHANGING TCB,
1190 00602 4775 JMS      GN5    /\GET MS NODE
1191 00603 3410 DCA I    X0     /\CLEAR SENDERWORD,
1192 00604 1010 TAD      X0     /\
1193 00605 3154 DCA      INTO   /\PTR TO MS[0].
1194 00606 3410 DCA I    X0     /\CLEAR LINKWORD.
1195 00607 1164 TAD      CURTSK /\
1196 00610 3410 DCA I    X0     /\PTR TO TCB[5] IN MS
1197 00611 1374 TAD      (MTFTCH-1/\
1198 00612 4571 JMS I    ZSNDREP /\SEND MS TO MTFTCH,
1199 00613 2164 ISZ      CURTSK /\
1200 00614 7350 AC3777    /\
1201 00615 0564 AND I    CURTSK /\CLEAR ONDISK BIT,
1202 00616 3564 DCA I    CURTSK /\
1203          /      JMP      SCED  /\

```

```

1204          /SCHEDULER.
1205 00617 7201 SCED, CLA IAC          /-?
1206 00620 3166          DCA          SCDREQ /-?SCDREQ NO LONGER PENDING.
1207 00621 6001          ION          /-?
1208 00622 3164 SCEDLP, DCA          CURTSK
1209 00623 3167          DCA          SCDINH /SCHEDULING NO LONGER INHIBITED,
1210 00624 1565          TAD I        HPRI0 /INSPECT PRI0 RUNQS,
1211 00625 7440 SCDL2, SZA          /FCHEX FALLS IN HERE,
1212          /NOTE: POSSIBLE CHANGES IN HPRI0 WILL
1213          /KICK US BACK TO SCED.
1214 00626 5233          JMP          SCDF /WE FOUND A TASK,
1215 00627 6202          CIF 0          /\DEAF WHILE UPDATING HPRI0!
1216 00630 2165          ISZ          HPRI0 /\UPDATE HPRI0 TO NEXT PRI0 RUNQ.
1217 00631 2165          ISZ          HPRI0 /\
1218 00632 5222          JMP          SCEDLP
1219 00633 3010 SCDF, DCA          X0
1220 00634 1010          TAD          X0
1221 00635 2167          ISZ          SCDINH /NO SCHEDULING WHILE BUILDING TASKS
1222 00636 3164          DCA          CURTSK /STATUS, STORE PTR TO TCB[5],
1223 00637 3051          DCA          CDFUF /SIGNAL FOR TSKSAF AND FCHEX ROUTINE,
1224 00640 1410          TAD I        X0 /B0=1: CODE OF TSK STILL ON DISK.
1225 00641 7710          SPA CLA
1226 00642 5200          JMP          ONDISK
1227 00643 1410          TAD I        X0
1228 00644 3107          DCA          BASE /TEMP
1229 00645 1107          TAD          BASE
1230 00646 4773 JMS          FCHECK /ASSERT VFLD IN CORE, IF NOT ISSUE
1231          /SWAPREQ IF POSSIBLE AND RETURN TO
1232          /SCEDNI,
1233
1234          IFDEF PDP8E <
1235          IFDEF EAE <
1236 00647 0127          AND          C70 /CDF TO ACT DF IN AC.
1237 00650 7110          CLL RAR;RTR
1238 00651 7012
1239 00652 3007          DCA          MONFL
1240 00653 1145          TAD          C7700
1241 00654 0107 PTBASE, AND          BASE /GET VFLDS AGAIN. LOC PTS AT BASE,
1242 00655 3163          DCA          VCURIF /CURRENT VINSTRFLD,
1243 00656 1163          TAD          VCURIF
1244 00657 7002          BSWR
1245 00660 4773 JMS          FCHECK
1246 00661 3321          DCA          CDIFI /STORE CDF TO VINSTRFLD.
1247 00662 1321          TAD          CDIFI
1248 00663 0127          AND          C70
1249 00664 1007          TAD          MONFL
1250 00665 3007          DCA          MONFL
1251 00666 1410          TAD I        X0
1252 00667 3004          DCA          MONAC
1253 00670 1410          TAD I        X0
1254 00671 3054          DCA          MONPC
1255 00672 1410          TAD I        X0 /GET MQ
1256 00673 7431          SWAB /LOAD MQ, ASSERT MODE B.
1257 00674 1410          TAD I        X0 /EAE +L
1258 00675 3110          DCA          X /TEMP.

```

```

1259 00676 1110 TAD X
1260 00677 7403 ACS
1261 00700 1133 TAD C200
1262 00701 0110 PTX, AND X /LOC PTS AT X.
1263 00702 7650 SNA CLA /MODE A OR B?
1264 00703 7447 SWBA
1265 00704 1110 TAD X /L+GTFL+UF+SCDINH TO FLAGS.
1266 00705 0145 AND C7700
1267 00706 1007 TAD MONFL
1268 00707 3007 DCA MONFL
1269 00710 1131 TAD C100 /USERMODE?
1270 00711 0007 AND MONFL
1271 00712 7650 SNA CLA
1272 00713 5316 JMP SCDF2 /SET BASE, X AND REQCDF.
1273 00714 1372 TAD (ISZ CDFUF/SET CDFUF ROUTINE TO SKIP.
1274 00715 5330 JMP SCDF3 /AND DONT TOUCH ANY LOCS IN USER'S FIELD.
1275 00716 1410 SCDF2, TAD X0
1276 00717 3150 DCA MON0
1277 00720 1410 TAD X0
1278 00721 6201 CDIFI, CDF /CDF TO VINSTRFLD.
1279 00722 3701 DCA PTX
1280 00723 1150 TAD MON0
1281 00724 3654 DCA PTBASE
1282 00725 1037 TAD XVCALL /REINITIALIZE VVRCDF FOR A NEW TASK.
1283 00726 3771 VRCDF3, DCA (VVRCDF+2/B000H! VRCDF FALLS IN HERE,
1284 00727 1321 TAD CDIFI
1285 00730 3052 SCDF3, DCA CDFUF+1 /NOT SURE ABOUT DF!
1286
1287 /FALL INTO MONITOR RETURN.
1288 00731 6203 MONEX, CIF CDF 0 /-MAKE SURE ABOUT DF.
1289 00732 3770 DCA EXTALL /-FSTEXT ALLOWED AGAIN.
1290 00733 1166 TAD SCDFREQ /-SCDFREQ PENDING?
1291 00734 7650 SNA CLA /-
1292 00735 5767 JMP TSXSAF /-YES; GO SCHEDULING.
1293 00736 1007 TAD MONFL /-JUST A SHORT RETURN.
1294 00737 6005 RTF /-
1295 00740 0377 AND (400 /-OVERRIDE PRIO SCHEDULING?
1296 00741 3167 DCA SCDINH /-ADJUST SCDINH,
1297 00742 1004 TAD MONAC /-
1298 00743 5454 JMP MONPC
1299 >
1300 >
1301
1302 IFDEF PDP8I <
1303 IFNDEF EAE <
1304 DCA CDFINS /CDF TO USERS' DF.
1305 TAD C7700
1306 PTBASE, AND BASE /GET VFLDS AGAIN. LOC PTS AT BASE.
1307 DCA VCURIF /CURRENT VIRTUAL INSTRFLD.
1308 TAD VCURIF
1309 BSWR
1310 JMS FCHECK
1311 DCA CDIFI /STORE CDF TO USERS' INSTRFLD.
1312 TAD CDIFI
1313 IAC

```

```

1314          DCA      CIFINS  /SET CIFINS TO USERS' IF.
1315          TAD      X0
1316          DCA      MONAC
1317          TAD      X0
1318          DCA      MONPC
1319          TAD      X0
1320          STORMQ    /TEMP
1321          TAD      X0
1322          DCA      MONFL
1323          TAD      C100      /USERMODE?
1324          AND      MONFL
1325          SNA CLA
1326          JMP      SCDF2     /NO, SET BASE, X, AND REQCDF.
1327          TAD      (1SZ CDFUF/SET CDFUF ROUTINE TO SKIP
1328          JMP      SCDF3     /AND DONT TOUCH ANY LOCS IN USERS' FIELD.
1329 SCDF2,      TAD      X0
1330          DCA      MON0
1331          TAD      X0
1332 CDIFI,      CDF      /TO USERS' INSTRFLD.
1333          DCA      (X
1334          GETMQ
1335          DCA      (MQ
1336          TAD      MON0
1337          DCA      PTBASE
1338          TAD      XVCALL    /REINITIALIZE VVRCDF FOR A NEW TASK.
1339 VRCDF3,     DCA      (VVRCDF+2/BOOH! VRCDF FALLS IN HERE.
1340          TAD      CDIFI
1341 SCDF3,      DCA      CDFUF+1 /NOT SURE ABOUT DF!
1342
1343          /FALL INTO MONITOR RETURN.
1344
1345 MONEX,      CIF CDF 0      /-MAKE SURE ABOUT DF.
1346          DCA      EXTALL   /-FAST EXIT ALLOWED AGAIN.
1347          TAD      SCDREQ    /-SCDREQ PENDING?
1348          SNA CLA          /-
1349          JMP      TSKSAF    /-YES; GO SCHEDULING.
1350          TAD      MONFL     /-
1351          AND      (400      /-
1352          DCA      SCDINH    /-ADJUST SCDINH FLAG.
1353 CIFINS,     CIF          /-TO USERS' IF
1354 CDFINS,     CDF          /-TO USERS' DF
1355          TAD      MONFL     /-
1356          RAL          /-SET LINK.
1357          AND      C200      /-USERMODE?
1358          SZA CLA          /-
1359          SUF          /-
1360          TAD      MONAC     /-
1361          ION          /-
1362          JMP      MONPC
1363          >
1364          >
1365
1366
1367          /IF CURRENT TASK IS PROTECT (USERMODE ON) THEN SKIP,
1368          /ELSE CDF CURRENT INSTRFLD.

```

/APP. A1. MC8,1. PAGE 0, INTR SECTION PAL8-V9H 02/14/77 PAGE 17-3

```
1369          /CALLED EITHER DEAF OR WITH SCDINH >0.  
1370          /CDFUF, 0          /!\?  
1371          /      ISZ      CDFUF  /!\?  
1372          /      JMP      CDFUF  /!\?  
1373          /WE PATCHED CDFUF ROUTINE OVER VCDIF ROUTINE.
```

```

1374      /FCHECK FAILED. VFLDS NOT IN CORE.
1375      /IF CALLED FROM A TSK (CDFUF 'NE' 0),
1376      /SAVE IT WITH THE NEW DF. OTHERWISE (CALLED FROM
1377      /SCHEDULER) SCHEDULE THE NEXT IMPORTANT TSK.
1378      /IF CURTSK IS REENTRANT (B1 OF TCB[6] SET), DONT SCHEDULE OTHER
1379      /TSKS OF THIS PRIORITY.
1380      00744 1164 FCHEX, TAD CURTSK
1381      00745 3010      DCA X0 /PTR TO TCB[6].
1382      00746 1051      TAD CDFUF /CALLED FROM TASK?
1383      00747 7650      SNA CLA
1384      00750 5354      JMP SCEDNI /NO.
1385      00751 2010      ISZ X0
1386      00752 1153      TAD MON3 /GET REQUESTED DATAFLD.
1387      / TAD (-FLDTAB /FLDTAB AT BEGIN OF PAGE?
1388      00753 5766' JMP MONSF2 /JMP INTO SAVE ROUTINE.
1389
1390      /WE CANNOT START UP THE FIRST TASK IN THIS RUNQ, BECAUSE ITS
1391      /VFLDS ARE NOT IN ACTUAL CORE.
1392      /TAKE THE NEXT ONE IN THIS RUNQ, UNLESS IT STARTS
1393      /WITH A REENTRANT TASK.
1394      00754 3167 SCEDNI, DCA SCDINH
1395      00755 1166      TAD SCDREQ /IF SCHEDULE REQ PENDING
1396      00756 7650      SNA CLA /JUMP RIGHT INTO THE SCHEDULER
1397      00757 5217      JMP SCED
1398      00760 7332      AC2000 /CURTSK REENTRANT
1399      00761 0410      AND I X0
1400      00762 7650      SNA CLA /IF SO, SKIP OVER THE REST OF THIS RUNQ.
1401      00763 1564      TAD CURTSK
1402      00764 5225      JMP SCDLP2
1403
1404
1405      IFDEF PDP8I <AVPT;(0;AVPT=-2>
1406      00766 0510
1407      00767 0554
1408      00770 0422
1409      00771 0022
1410      00772 2051
1411      00773 2000
1412      00774 5050
1413      00775 4346
1414      00776 0545
1415      00777 0400
1416      1000 PAGE

```

```

1417          /***** SOME SMALL DEAF ROUTINES USED BY THE SCHEDULER *****/
1418
1419          /SEND.
1420          /INSERT A MS IN TSKS RCQ. IF TSK WAS WAITING FOR MSS FROM
1421          /THIS PROCES, INSERT IT IN RUNQ (BY CALLING INRUNQ).
1422
1423          /INT0=PTR TO MS[0] SENDERWORD.
1424          /AC =PTR TO TCB[4] TSK WAITWORD.
1425          /USES INT0,INT2.
1426
1427          /REPORT.
1428          /INSERT A MS IN TSKS RPQ. IF TSK WAS WAITING FOR THAT MS INSERT
1429          /TSK IN RUNQ (BY CALLING INRUNQ).
1430          /
1431          /INT0 =PTR TO MS[0] SENDERWORD.
1432          /USES INT1,INT2.
1433          01000 0000      SNDREP, 0          /\
1434          01001 7100      CLL                /\
1435          01002 7440      SZA                /\
1436          01003 5206      JMP      .+3        /\
1437          01004 7240      CLA CMA            /\
1438          01005 1554      TAD      INT0      /\CML
1439          01006 3155      DCA      INT1      /\
1440          01007 7420      SNL                /\SKIPS IF REPORT
1441          01010 7344      ACM2              /\
1442          01011 1136      TAD      M2        /\
1443          01012 1155      TAD      INT1      /\
1444          01013 3156      DCA      INT2      /\PTR TO TCB[4] IF SEND
1445                                          /\PTR TO TCB[2] IF REPORT
1446                                          /\L=0 IF REP, L=1 IF SND
1447          01014 7010      RAR                /\AC=0; REP, AC4000; SND.
1448          01015 7051      CIA RAR            /\AC=4000; REP, AC=2000; SND.
1449          4000      FWTRP=4000;FWTMS=2000    /\
1450          2000
1451          01016 0555      AND      INT1      /\
1452          01017 7450      SNA                /\WAITING FOR SEND OR REPORT?
1453          01020 5251      JMP      MSAPP      /\NO. ONLY APPPEND.
1454          01021 4777      JMS      CHKMS      /\MS ALLOWED TO REACTIVATE TSK?
1455          01022 5251      JMP      MSAPP      /\
1456          01023 7307      MSIN, AC4          /\
1457          01024 1155      TAD      INT1      /\
1458          01025 3156      DCA      INT2      /\PTR TO TSKS AC.
1459          01026 1155      TAD      INT1      /\
1460          01027 7040      CMA                /\
1461          01030 1164      TAD      CURTSK     /\REPORT MSG TO CUR TSK?
1462          01031 7640      SZA CLA          /\
1463          01032 5235      JMP      .+3        /\
1464          01033 1376      TAD      (MONAC     /\NOTE ONLY HERE IF CURTSK IS WAITING
1465                                          /\FOR THIS REPORT AND YET ITS STATUS
1466                                          /\ IS NOT SAVED!
1467          01034 3156      DCA      INT2      /\PTR TO TASKS AC
1468          01035 7326      AC2                /\
1469          01036 1154      TAD      INT0      /\
1470          01037 3556      DCA      INT2      /\MSPTR IN TASKS AC.
1471          01040 4276      JMS      INRUNQ     /\INSERT IN RUNQ.

```

```

1472 01041 7325 AC3 /!\
1473 01042 1156 TAD INT2 /!\NOTE MONFL=MONAC+3!
1474 01043 3156 DCA INT2 /!\
1475 01044 7330 AC4000 /!\
1476 01045 0555 AND I INT1 /!\0 IF SND, 4000 IF REPORT
1477 01046 1556 TAD I INT2 /!\
1478 01047 3556 DCA I INT2 /!\SET TSKS L IF REPORT
1479 01050 5600 JMP I SNDREP /!\
1480
1481 01051 2156 MSAPP, ISZ INT2 /!\PTR TO HEAD OF MSQ,
1482 01052 2154 ISZ INT0 /!\
1483 01053 1554 TAD I INT0 /!\MS ALREADY IN SOME Q?
1484 01054 7640 SZA CLA /!\IF AC=0, PROBABLY NOT.
1485 01055 5600 JMP I SNDREP /!\YES. LEAVE, DONT MAKE CIRCULAR QS,
1486 IFDEF STATX <
1487 01056 1275 TAD MSQMAX /!\SET UP COUNTER.
1488 01057 7040 CMA /!\
1489 01060 3155 DCA INT1 /!\CTR
1490 >
1491 01061 1556 MSAPP1, TAD I INT2 /!\SEARCH MSQTAIL,
1492 01062 7450 SNA /!\
1493 01063 5271 JMP MSAPP2 /!\
1494 01064 3156 DCA INT2 /!\
1495 IFDEF STATX <
1496 01065 2155 ISZ INT1 /!\INCREMENT CTR.
1497 01066 5261 JMP MSAPP1 /!\
1498 01067 2275 ISZ MSQMAX /!\IF OVERFLOW INCREMENT QMAX,
1499 >
1500 01070 5261 JMP MSAPP1 /!\
1501 01071 1154 MSAPP2, TAD INT0 /!\APPEND MESSAGE
1502 01072 3556 DCA I INT2 /!\
1503 01073 3554 DCA I INT0 /!\WARNING: IT SEEMS VERY ROTTEN FOR
1504 /!\INTMS!!! BUT RPQ WON'T BECOME CIRCULAR,
1505 /!\AS WE ONLY COME HERE IF INTMS WAS THE
1506 01074 5600 JMP I SNDREP /!\LAST ONE IN RPQ!!!!
1507
1508 01075 0000 IFDEF STATX <MSQMAX, 0 >
1509

```

```

1510      /THE RUNQS.
1511      /THE SCHEDULER USES A PRIORITY RUNQ, I.E. A RUNQ FOR EACH
1512      /PRIORITY. EACH TSK HAS A PRIORITY IN THE RANGE 1-7. PRIORITY 0
1513      /IS RESERVED FOR MONTSKS, PRIORITY 10 FOR THE IDLETSK,
1514      /THE HIGHEST PRIORITY IS PRIO0.
1515      /THE ARRAY RQPTRS HOLDS THE HEADS AND TAILS OF EACH RUNQ AS
1516      /FOLLOWS:
1517      /      RQPTRS[2N] HEAD OF RUNQ OF PRIO N.
1518      /      RQPTRS[2N+1] TAIL OF RUNQ OF PRIO N.
1519      /THE ITEMS IN THE RUNQ ARE TCBS OF RUNNABLE TSKS. THEY ARE
1520      /THREADED THROUGH THE TCB LINKWORDS (TCB[5]).
1521      /IN PRINCIPLE EACH PRIO RUNQ IS A FIFO Q, BUT FCHEX
1522      /MAY SKIP OVER TASKS WHOSE FLDS ARE NOT IN ACTUAL CORE.
1523
1524      /INRUNQ.
1525      /ERASE ALL WAIT CONDITIONS, BUT NOT STOPD.
1526      /INSERT A TSK IN ITS PRIO RUNQ, IF IT IS NOT STOPPED (STOPDBIT
1527      /SET). SIGNAL SCEDREQ BY CLEARING SCEDREQ.
1528      /
1529      /INT1 =PTR TO TCB[4] WAIT WORD.
1530      /USES INT0, INT2.
1531      INRUNQ, 0      /\
1532      01076 0000      TAD      INT1      /\REMOVE ANY WAIT CONDITIONS EXCEPT
1533      01077 1555      AND      C37      /\STOP
1534      01100 0125      DCA      INT1      /\
1535      01101 3555      TAD      INT1      /\2*PRIO
1536      01102 1555      CLL CMA RAR      /\STOPDBIT TO LINK,
1537      01103 7150      SNL                      /\STOPPDBIT SET?
1538      01104 7420      JMP      INRGEX /\YES. SKIP THE REST, AC>=0.
1539      01105 5327      STL CMA RAL      /\GET PRIO*2+1
1540      01106 7164      TAD      (RQPTRS /\
1541      01107 1375      DCA      INT2      /\PTR TO TAIL OF RUNQ.
1542      01110 3156      ISZ      INT1      /\INT1:=PTR TO TCB[5] LINK WORD.
1543      01111 2155      TAD      INT1      /\GET PTR IN TIMEOUT Q IF ANY.
1544      01112 1555      DCA      INT0      /\PTR IN TOQ OR INNOCENT PTR.
1545      01113 3154      DCA      INT0      /\CLEAR POSSIBLE TIMEOUTNODE.
1546      01114 3554      TAD      INT2      /\PREVIOUS LAST ITEM.
1547      01115 1556      DCA      INT0      /\PTR TO PRECESSOR.
1548      01116 3154      TAD      INT1      /\
1549      01117 1155      DCA      INT0      /\APPEND.
1550      01120 3554      DCA      INT1      /\ZERO TCB[5].
1551      01121 3555      TAD      INT1      /\UPDATE TAIL.
1552      01122 1155      DCA      INT2      /\
1553      01123 3556      TAD      HPRIO /\COMPARE NEW PRIO TO CURRENT HIGHEST.
1554      01124 1165      CIA                      /\=HPRIO
1555      01125 7041      TAD      INT2 /\=HPRIO+NPRI0+1
1556      01126 1156      INRGEX, SMA CLA /\MUST WE SET SCDEQ?
1557      01127 7700      JMP      INRUNQ /\NO.
1558      01130 5676      CMA                      /\
1559      01131 7040      TAD      INT2 /\YES. SET NEW HPRI0.
1560      01132 1156      DCA      HPRI0 /\
1561      01133 3165      DCA      SCDEQ /\
1562      01134 3166      JMP      INRUNQ /\
1563      01135 5676

```

```

1563      /OURUNQ.
1564      /REMOVE A TASK FROM THE RUNQ.
1565      /SET UP WAITCONDITIONS IN TSK WAITWORD TCB[4].
1566      /WAITCONDITIONS:
1567      4000 FWTRP= 4000      /WAIT REPORT.
1568      2000 FWTMS= 2000      /WAIT FOR MS TO BE RECEIVED.
1569      0200 FWTTM= 200      /WAIT FOR TIME OUT.
1570      0400 FWTSWP= 400      /WAIT FOR SWAP IN.
1571      0001 STOPD=1 /STOPPED. WAITING FOR RESUME COMMAND.
1572      /INT3 CONTAINS WAITCONDITIONS.
1573      /AC=PTR TO TCB[4] TSK WAITWORD.
1574      /USES INT1,INT2,INT3.
1575      /
1576      01136 0000 OURUNQ, 0      /\
1577      01137 3155      DCA      INT1      /\PTR TO TCB[4].
1578      01140 1555      TAD      INT1      /\FETCH 2*PRIO.
1579      /      AND      (36      /\REMOVE WAITS?
1580      01141 1375      TAD      (RQPTRS /\
1581      01142 3156      DCA      INT2      /\PTR TO PRIO RUNQ.
1582      01143 1157      TAD      INT3      /\GET WAITCONDITIONS.
1583      01144 1555      TAD      INT1      /\SET UP WAIT CONDITIONS.
1584      01145 3555      DCA      INT1      /\
1585      01146 2155      ISZ      INT1      /\PTR TO TCB[5] LINK WORD.
1586      01147 1156      TAD      INT2      /\INITIATE SEARCH FOR PRECESSOR.
1587      01150 3157 OURQLP, DCA      INT3      /\PTR IN Q.
1588      01151 1557      TAD      INT3      /\FETCH LINK
1589      01152 7041      CIA      /\
1590      01153 1155      TAD      INT1      /\WHO PTS AT THE TSK TO BE REMOVED?
1591      01154 7650      SNA CLA      /\
1592      01155 5360      JMP      OURQF      /\PRECESSOR FOUND.
1593      01156 1557      TAD      INT3      /\NOT FOUND YET, SEARCH DOWN THE Q.
1594      01157 5350      JMP      OURQLP /\
1595      01160 1555 OURQF, TAD      INT1      /\DELETE TSK FROM RUNQ.
1596      01161 3557      DCA      INT3      /\
1597      01162 1155      TAD      INT1      /\
1598      01163 3555      DCA      INT1      /\INNOCENT PTR IN LINKWORD (SEE INRUNQ).
1599      01164 1557      TAD      INT3      /\IS THE REMOVED TSK THE LAST ONE OF THE
1600      01165 7640      SZA CLA      /\Q
1601      01166 5736      JMP      OURUNQ /\NO.
1602      01167 2156      ISZ      INT2      /\YES; UPDATE TAIL. INT2:=PTR TO TAIL.
1603      01170 1157      TAD      INT3      /\NEW LAST TSK.
1604      01171 3556      DCA      INT2      /\
1605      01172 5736      JMP      OURUNQ /\
1606      /NOTE: QS CONTAINING ONLY ONE TSK ARE TREATED WELL THIS WAY!
1607
1608      IFNDEF STATX < AVPT;(0;AVPT=-2 >
1609      01175 4700
1610      01176 0004
1611      01177 3514
1612      1200 PAGE

```

```

1613      /APP. A2. MC8.2. VIRTUAL CORE MANAGER, PAGE- AND FIELDREQUEST.
1614      /***** VIRTUAL CORE MANAGER *****/
1615      /
1616      /THIS OPERATING SYSTEM IS EQUIPED WITH UPTO 64 VIRTUAL MEMORY
1617      /FIELDS. THE EXACT NUMBER OF VIRTUAL FIELDS IS CONTROLLED BY THE
1618      /PARAMETER VFMAX IN THE CONFIG FILE.
1619      /EACH VIRTUAL FIELD MAY BE SWAPPED INTO AN ACTUAL FIELD OF THE
1620      /PDP8. THE NUMBER OF ACTUAL FIELDS IS DETERMINED BY THE PARAMETER
1621      /ACTMAX IN THE CONFIG FILE.
1622      /IF A VIRTUAL FIELD IS NOT IN CORE, IT IS STORED ON THE DISK IN
1623      /ITS OWN FIELDSLOT (A SPECIFIC AREA ON THE DISK).
1624      /
1625      /THE STATUS OF A VIRTUAL FIELD IS SPECIFIED IN THE FIELDTABLE
1626      / (FLDTAB). EACH VIRTUAL FIELD OCCUPIES ONE WORD IN FLDTAB.
1627      /BIT ASSIGNMENT OF FLDTAB:
1628      /B0: 0: ON DISK, 1: IN CORE; SEE BELOW.
1629      /B1: 0: NEVER USED, 1: USED (STATISTICS).
1630      /B2: GIVEN OUT AS WHOLE FIELD FOR DATA; GENERAL PAGE 0
1631      / ROUTINES CANNOT BE COPIED INTO THIS FIELD.
1632      1000 FDATA=1000
1633      /B3: 0: PART OF PAGE 0 OF THIS FIELD IS OCCUPIED.
1634      0400 FZREQ=400
1635      /B5: CORERESIDENT; NOT ALLOWED TO SWAP THIS FIELD OUT OF CORE.
1636      0100 FCRES=100
1637      /B6-8 ACTUAL FIELD IF THE FIELD IS IN CORE.
1638      /B9: MONIN BIT; A COPY OF THE GENERAL PAGE 0 ROUTINES IS IN
1639      / THIS FIELD.
1640      0004 FMONIN=4
1641      /B10: 0: NO DATA PRESENT, 1: DATA PRESENT (USED TO
1642      / OPTIMIZE SWAPPING).
1643      0002 FDP=2
1644      /B11: 0: FREE, 1: OCCUPIED.
1645      0001 FOCCUP=1
1646      /
1647      /NOTES:
1648      / IF A FIELD IS IN CORE AND DATA ARE PRESENT, WE COMPUTE A CDF
1649      /TO ITS ACTUAL FIELD BY ANDING THE BITS IN FLDTAB WITH 6271.
1650      /THIS TRIC FAILS IF THE FIELD IS FREE!!!.
1651      / SWAPPING IN OR OUT A FIELD IS A DUMMY ACTION IF THE DATA
1652      /PRESENT BIT (B10) IS LOW.
1653      /
1654      /TWO DIFFERENT STORAGE ALLOCATION SYSTEMS ARE USED TO ALLOCATE
1655      /STUFF IN THESE VIRTUAL FIELDS.
1656      /THE ROUTINES GETFLD AND FRFLD ARE USED TO REQUEST AND RETURN
1657      /COMPLETE FIELDS.
1658      /THESE ROUTINES DIRECTLY ACCESS THE FIELDTABLE TO CHECK, SET AND
1659      /CLEAR THE NONFREE BIT (B11).
1660      /A NUMBER OF UPTO 37 PAGES CAN BE REQUESTED AND RETURNED BY
1661      /CALLING THE ROUTINE PGRQN. IT USES A CHAIN OF FREE CORE JUNKS
1662      / (FCCHAN). ON REQUEST PGRQN TRIES TO FIND A JUNK OF SUFFICIENT
1663      /LENGTH IN FCCHAN. IF THIS FAILS GETFLD IS CALLED. LIKEWISE ON
1664      /RETURN FRFLD IS CALLED IF A FREE CORE JUNK OF 37 PAGES IS
1665      /RETURNED. PAGE 0 IS NEVER GIVEN OUT BY PGRQN. IT IS RESERVED
1666      /FOR THE GENERAL PAGE 0 ROUTINES. NONPROTECT TASKS (USERMODE
1667      /OFF) HAVE A MAXIMUM LENGTH OF 37 PAGES AND USE THE GENERAL

```

1668 /PAGE 0. THEY ARE ALLOCATED USING PGRQN,
1669 /A TASK HAS ACCESS TO AT MOST TWO DIFFERENT FIELDS (IF AND DF).
1670 / (NOTE: WE DIDNOT ALLOW THE CIF INSTRUCTION IN A TASK).
1671 /BEFORE RUNNING A TASK THE ROUTINE FCHECK CHECKS WETHER THESE
1672 /FIELDS ARE ACTUALLY IN CORE. IF A TASK WANTS TO CHANGE ITS
1673 /DATAFIELD IT CALLS EITHER VVCDF OR VVRCDF. THESE ROUTINES CHECK
1674 /WETHER THE NEW DATAFIELD IS IN CORE.
1675 /
1676 /SWAPPING FIELDS.
1677 /IF SOMEONE WANTS TO ACCESS A FIELD WHICH IS NOT IN CORE AT THAT
1678 /MOMENT, IT MUST BE SWAPPED IN. BEFORE THAT ANOTHER FIELD MUST BE
1679 /SWAPPED OUT. WHICH FIELD IS TO BE REMOVED IS DETERMINED BY THE
1680 /PAGING ROUTINES. WHICH FIELDS ARE ACTUALLY IN CORE IS DENOTED IN
1681 /CORMAP (COREMAP). IT IS AN ARRAY OF ACTMAX ENTRIES, AN ENTRY FOR
1682 /EACH ACTUAL FIELD.
1683

```

1684      /***** PAGE REQUEST AND RETURN SECTION.
1685      /
1686      /1-37 PAGES OF FREE CORE MAY BE REQUESTED BY CALLING THE ROUTINE
1687      /PGRQN.
1688      /JUNKS OF FREE CORE ARE ORGANIZED IN A CHAIN; FCCHAN (FREE CORE
1689      /CHAIN). THE CHAIN IS SORTED ACCORDING TO INCREASING PAGE AND
1690      /FLDNUMBER. WHEN A CERTAIN AMOUNT OF PAGES IS REQUESTED WE FIRST
1691      /SEARCH THE CHAIN TO FIND A JUNK OF SUFFICIENT LENGTH. IF THE
1692      /SEARCH FAILS WE CALL GETFLD TO FETCH A FRESH EMPTY FLD. THIS IS
1693      /BROKEN INTO TWO JUNKS (NOT IF 37 PAGES WERE REQUESTED), ONE JUNK
1694      /IS HANDED TO THE CALLER, THE OTHER IS INSERTED IN THE CHAIN.
1695      /PAGE 0 OF EACH FLD IN THIS SYSTEM IS RESERVED FOR THE MONITOR.
1696      /
1697      /PAGES PREVIOUSLY REQUESTED BY CALLING PGRQN MUST BE RETURNED BY
1698      /CALLING PGRQN. THE NEW JUNK IS INSERTED
1699      /IN THE CHAIN POSSIBLY MERGED WITH OTHER JUNKS. IF A WHOLE FLD
1700      / (37 PAGES) IS FREE IT IS NOT INSERTED IN THE CHAIN.
1701      /FRFLD IS CALLED TO SET THE EMPTY CONDITION IN FLDTAB.
1702      /
1703      /LAYOUT OF A NODE IN THE FREE CORE CHAIN:
1704      /W0: PTR TO NEXT NODE; 0 INDICATES THE END.
1705      /W1: B0-5 VFLD#, B7-11 FIRST FREE PAGE.
1706      /W2: MINUS NUMBER OF FREE PAGES.
1707      /
1708
1709      /WE HOPE THESE ARE INNOCENT TEMPS.
1710      0070 PGRTEM=VJUMS
1711      0045 PGRREM=VVRDF
1712      0025 PGRLEN=VVCDF
1713      0020 PGROLD=VVRCDF
1714
1715      /HELP ROUTINE FOR PAGE RETURN AND REQUEST.
1716      /THIS ROUTINE IS CALLED WHEN INITIALIZING PGRQ AND PGRTN.
1717      /THE ROUTINE IS REENTERED HALFWAY EACH TIME A NEW NODE HAS
1718      /TO BE INSPECTED.
1719      /      JMS      PGR
1720      /      SNA              /AC=0 IF END OF CHAIN REACHED.
1721      /      JMP      EXH      /FCCHAN EXHAUSTED.
1722      /      ///
1723      /LENGTH IN MARG1 B7-11.
1724      /AT RETURN AC CONTAINS FIRST ITEM OF NEXT NODE.
1725
1726      01200 0000 PGR,      0
1727      01201 1161      TAD      MARG1      /GET LENGTH.
1728      01202 0125      AND      C37      /REMOVE OPTIONS,
1729      IFDEF CHECK      <
1730      01203 7450      SNA
1731      01204 4576      SYSPLT      /DONT ACCEPT ZERO LENGTH
1732      >
1733      01205 3025      DCA      PGRLEN      /LENGTH
1734      01206 1377      TAD      (FCCHAN      /FETCH PTR TO HEAD OF FREE CORE CHAIN,
1735      01207 5212      JMP      .+3      /FALL INTO LOOP.
1736      01210 7200 PGRLP,  CLA      /NEXT NODE. ROUTINE IS REENTERED HERE.
1737      01211 1420      TAD      PGROLD      /UPDATE PTR TO PRECESSOR.
1738      01212 3020      DCA      PGROLD      /STORE PTR TO PRECESSOR.

```

1739	01213	1420	PGR LPM, TAD	I	PGR OLD	/JUMP HERE AFTER A MERGE.
1740	01214	7450			SNA	/LIST EXHAUSTED?
1741	01215	5600	PGR EXH, JMP	I	PGR	/RETURN WITH AC=0.
1742	01216	3010			DCA	/PTR TO NODE TO BE INSPECTED.
1743	01217	1410			TAD	/FIRST ITEM= FLD+PAG, NEVER 0!
1744	01220	5600			JMP	
1745						
1746						

```

1747      /PGRQN, PAGE REQUEST/RETURN,
1748      /REQUEST: AC=0, RETURN: AC=FLD+PAG.
1749      01221 0000 PGRQN, 0
1750      01222 7440      SZA      /REQUEST OR RETURN?
1751      01223 5266      JMP      PGRTN
1752      /PGRQ, PAGE REQUEST.
1753      /REQUEST 1-37 PAGES OF FREE CORE.
1754      /OPTIONS:      CRES      REQUEST PAGES IN CORE RESIDENT FLD.
1755      /              ZREQ      REQUEST TO USE THE OTOPART OF PAGE 0 IN
1756      /              FLD OF THESE PAGES.
1757      /              JMS      PGRQN
1758      /OPTIONS+LENGTH IN MARG1.
1759      /AC=FLD+PAG AT RETURN. B0-4 PAGE#, B6-11 VFDL#.
1760      /USES X0, MON3, MARG1, MONAC.
1761      /
1762      PGRQ,
1763      01224 4200      JMS      PGR      /FALL INTO INSPECT NODES LOOP.
1764      01225 7450      SNA
1765      01226 5776      JMP      PGRQX      /JUMP FOR EXHAUSTED LIST.
1766      01227 3070      PGRQLP, DCA      PGRTEM      /STORE FLD+PAG,
1767      01230 1410      TAD      X0      /FETCH MINUS LENGTH
1768      01231 1025      TAD      PGRLEN      /COMPARE SIZE,
1769      01232 7540      SMA SZA
1770      01233 5210      JMP      PGRLP      /TO SHORT, NEXT NODE.
1771      01234 3045      DCA      PGRREM      /STORE MINUS REMAINDER.
1772      01235 1070      TAD      PGRTEM
1773      01236 4355      JMS      RDFLD      /MON3:=PTR IN FLDTAB, CONTENTS OF FLDTAB
1774      01237 0375      AND      (FCRES+FZREQ      /IN AC,
1775      /NOW A VERY COMPLICATED TEST FOR THE OPTIONS FOLLOWS.
1776      /THIS FLD AND THE OPTIONS MATCH IF:
1777      /NOT BOTH OF THEM HAS THE ZREQBIT SET
1778      /AND EITHER BOTH OR NONE HAS THE CRESBIT SET.
1779      01240 1161      TAD      MARG1      /NOTE: ZREQ AND CRES ARE NON-ADJACENT
1780      01241 0374      AND      (FZREQ+2+FCRES      /BITS.
1781      01242 7640      SZA CLA
1782      01243 5210      JMP      PGRLP      /NO MATCH.
1783      01244 1161      TAD      MARG1      /SET UP ZREQBIT IN FLDTAB IF NECESSARY.
1784      01245 0373      AND      (FZREQ
1785      01246 1553      TAD      MON3      /WE KNOW IT IS NOT SET YET.
1786      01247 3553      DCA      MON3
1787      01250 1420      TAD      PGROLD      /BREAK OR DELETE THIS NODE.
1788      01251 3010      DCA      X0
1789      01252 1070      TAD      PGRTEM      /FLD+PAG
1790      01253 1025      TAD      PGRLEN      /ADD LENGTH
1791      01254 3410      DCA      X0      /NEW FLD+PAG
1792      01255 1045      TAD      PGRREM      /MINUS NEW LENGTH
1793      01256 7450      SNA
1794      01257 4772      JMS      DF3      /BAD TRICK:
1795      01260 3410      DCA      X0      /FIRST DELETE AND FREE NODE.
1796      01261 1070      PGREX, TAD      PGRTEM      /AND THEN STILL CHANGE ITS LAST ITEM,
1797      01262 7112      CLL RTR;RTR;RTR
1798      01263 7012
1799      01264 7012
1800      01265 5621      JMP      PGRQN
1801

```

```

1802
1803 /PGRTN. PAGE RETURN.
1804 /RETURN PAGES OF FREE CORE PREVIOUSLY REQUESTED BY CALLING PGRQN.
1805 /THE NEW JUNK OF FREE CORE IS INSERTED IN THE FREE CORE CHAIN.
1806 /IF POSSIBLE IT IS MERGED WITH ADJACENT JUNKS. IF A WHOLE FLD IS
1807 /FREE AGAIN (37 FREE PAGES) IT IS REMOVED FROM THE CHAIN AND THE
1808 /EMPTY CONDITION IS SET IN FLDTAB.
1809 / TAD /FLD+PAG IN AC, B0-4 PAGE#, B6-11 VFLD#.
1810 / JMS PGRQN
1811 /LENGTH+FZREQBIT IN MARG1
1812 /
1813 PGRTN,
1814 01266 7106 CLL RTL;RTL;RTL
1815 01267 7006
1816 01270 7006
1817 01271 3045 DCA PGRREM /FLD+PAG.
1818 01272 1373 TAD (FZREQ
1819 01273 0161 AND MARG1
1820 01274 7040 CMA
1821 01275 3200 DCA PGR /SAVE THIS OPTION.
1822 01276 1045 TAD PGRREM
1823 01277 4355 JMS RDFLD /GET CONTENTS OF FLDTAB
1824 01300 0200 AND PGR /CLEAR POSSIBLE ZREQBIT.
1825 01301 3553 DCA MON3
1826 01302 3070 DCA PGRTEM /CLEAR PTR FOR RETURN.
1827 PGRTN2, /PGRQ MAY FALL IN HERE.
1828 01303 4200 JMS PGR /FALL INTO INSPECT NODE LOOP.
1829 01304 7450 SNA
1830 01305 5332 JMP PGRNM /JUMP FOR EXHAUSTED LIST. APPEND HERE.
1831 01306 7141 CIA CLL /AC:=MINUS FLD+PAG OF ITEM. NONZERO!
1832 01307 1045 TAD PGRREM /COMPARE WITH FLD+PAG TO BE INSERTED.
1833 01310 7420 SNL /INSERT IT HERE?
1834 01311 5327 JMP PGRMB /YES.
1835 01312 1410 TAD X0 /NO. PERHAPS FORWARD MERGE? ADD MINUS
1836 01313 7640 SZA CLA /LENGTH.
1837 /AC=FLD+PAG TO BE INSERTED
1838 /-(FLD+PAG +LENGTH) OF NODE.
1839 01314 5210 JMP PGRLP /NO FORWARD MERGE, NEXT NODE.
1840 01315 1420 PGRMF, TAD PGRCLO /FORWARD MERGE.
1841 01316 3010 DCA X0
1842 01317 1410 TAD X0 /UPDATE FLD+PAG TO BE INSERTED.
1843 01320 3045 DCA PGRREM
1844 01321 1410 PGRM2, TAD X0 /BACKWARD MERGE JUMPS HERE.
1845 01322 7041 CIA
1846 01323 1025 TAD PGRLEN /UPDATE LENGTH TO BE INSERTED.
1847 01324 3025 DCA PGRLEN
1848 01325 4772 JMS DF3 /DELETE AND FREE MERGED NODE.
1849 01326 5213 JMP PGRLOPM /WE WILL FIND THE PLACE TO INSERT
1850 /IMMEDIATELY.
1851 01327 1025 PGRMB, TAD PGRLEN /ADD LENGTH
1852 01330 7650 SNA CLA /AC=FLD+PAG+LENGTH TO BE INSERTED -
1853 /-(FLD+PAG) OF NODE.
1854 01331 5321 JMP PGRM2 /BACKWARD MERGE.
1855 01332 1371 PGRNM, TAD (-37 /NO MERGE.
1856 01333 1025 TAD PGRLEN /WHOLE FLD FREE?

```

1857	01334	7650	SNA CLA		
1858	01335	5351	JMP	PGRFF	/YES.
1859	01336	4770	JMS	GN3	/GET NODE. X0 PTS TO NODE AT RETURN.
1860	01337	1420	TAD	PGROLD	
1861	01340	3410	DCA	X0	
1862	01341	1010	TAD	X0	
1863	01342	3420	DCA	PGROLD	/INSERTED IN CHAIN.
1864	01343	1045	TAD	PGRREM	
1865	01344	3410	DCA	X0	
1866	01345	1025	TAD	PGRLEN	
1867	01346	7041	CIA		/MINUS LENGTH.
1868	01347	3410	DCA	X0	
1869	01350	5261	JMP	PGREX	
1870	01351	1045	PGRFF, TAD	PGRREM	/SET UP EMPTY CONDITION IN FLDTAB.
1871	01352	7002	BSWR		
1872	01353	4767	JMS	FRFLD	
1873	01354	5261	JMP	PGREX	
1874					
1875	01355	0000	RDFLD,	0	
1876	01356	7002	BSWR		
1877	01357	0130	AND	C77	
1878	01360	1112	TAD	XFLDTAB	
1879	01361	3153	DCA	MON3	
1880	01362	1553	TAD	MON3	
1881	01363	5755	JMP	RDFLD	
1882					
1883	01364	0560	AVPT;(0;AVPT=.	-2	
1884	01365	1366			
1885		1364			
1886					
1887	01366	0000			
1888	01367	1507			
1889	01370	4340			
1890	01371	7741			
1891	01372	4317			
1892	01373	0400			
1893	01374	1100			
1894	01375	0500			
1895	01376	1400			
1896	01377	4721			
1897		1400	PAGE		

1898	01400	1377	PGRQX,	TAD	(FCRES+PZREQ/LIST EXHAUSTED, GET A NEW EMPTY FLD.
1899	01401	0161		AND	MARG1
1900	01402	4247		JMS	GETFLD /SETS BITS IN AC IN FLDTAB.
1901					/RETURNS VFLD# IN AC B6-11.
1902	01403	7106		CLL	RTL;RTL;RTL
1903	01404	7006			
1904	01405	7006			
1905	01406	7001		IAC	/LEAVE PAGE 0 FOR MONITOR.
1906	01407	3070		DCA	PGRTEM
1907	01410	1553		TAD	MON3 /GET STATUS OF FLD.
1908					/IF IT IS IN CORE AND MONINBIT NOT
1909					/SET, WE MUST COPY THE GENERAL PAGE ZERO
1910					/INTO IT.
1911	01411	7510		SPA	/SKIP IF NOT IN CORE; AC 'NE' 0.
1912	01412	0122		AND	C4;MONIN=4
1913		0004			
1914	01413	7650		SNA	CLA
1915	01414	4776		JMS	GPZRIN /INSERT GENERAL PAGE ZERO STUFF,
1916	01415	1025		TAD	PGRLEN /COMPUTE LENGTH OF REMAINDER
1917	01416	7041		CIA	
1918	01417	1125		TAD	C37
1919	01420	7450		SNA	
1920	01421	5775		JMP	PGREX /NO REMAINDER
1921	01422	3161		DCA	MARG1
1922	01423	1070		TAD	PGRTEM
1923	01424	1025		TAD	PGRLEN /FLD+PAG OF REMAINDER
1924	01425	3045		DCA	PGRREM
1925	01426	5774		JMP	PGRTN2 /RETURN THIS JUNK AND LEAVE PGRON,
1926					

```

1927          /***** LOCKING AND UNLOCKING FIELDS.
1928          /
1929          /FLDS MAY BE LOCKED IN CORE FOR SEVERAL REASONS:
1930          /A.    THE TASKS IN THIS FLD ARE USED SO OFTEN THAT IT IS WORTH
1931          /      TO HAVE THEM CORERESIDENT (SEE THE USE OF THE CRESBIT IN
1932          /      THE PAGEREQUEST ROUTINES).
1933          /B.    ROUTINES CONNECTED TO AN INTSLOT MUST RESIDE IN
1934          /      CORERESIDENT FLDS.
1935          /C.    I/O PROCESSES MAY TEMPORARILY LOCK FLDS IN CORE (E.G.,
1936          /      DATABREAK).
1937          /FLDS ARE LOCKED (UNLOCKED) BY CALLING THE ROUTINE VFLOCK, WITH
1938          /AC=100 (AC=-100).
1939          /THEY ASSUME THE VFOLD TO BE LOCKED (UNLOCKED) TO BE PRESENT IN
1940          /CORE AND MON3 TO POINT TO ITS ENTRY IN FLDTAB.
1941          /LOCKS ARE COUNTED IN CORMAP 80-5.
1942          0100  FCRES=100          /CORERESIDENCYBIT.
1943
1944          01427  0000  VFLOCK, 0
1945          01430  3246          DCA      VFLTM2  /TEMP
1946          01431  1553          TAD      MON3
1947          01432  0127          AND      C70
1948          01433  7112          CLL RTR
1949          01434  7010          RAR
1950          01435  1170          TAD      ZCORMAP
1951          01436  3245          DCA      VFLTEM
1952          01437  1645          TAD      VFLTEM
1953          01440  1246          TAD      VFLTM2  /CHANGE LOCK COUNT.
1954
1955          IFDEF CHECK <
1956          01441  7510          SPA
1957          01442  4576          SYSHLT          /WE DONT ACCEPT MORE THAN 37 LOCKS.
1958          >
1959          01443  3645          DCA      VFLTEM
1960          01444  5627          JMP      VFLOCK
1961
1962          01445  0000  VFLTEM, 0
1963          01446  0000  VFLTM2, 0

```

```

1964          /***** GETFLD AND FRFLD,
1965          /
1966          /FIELDS ARE CLAIMED BY CALLING GETFLD. THEY ARE FREED BY CALLING
1967          /FRFLD. GETFLD SEARCHES THE FIRST FREE FIELD FROM THE FLDTAB AND
1968          /RETURNS ITS NUMBER IN AC B6-11, POSSIBLE SPECIAL CONDITIONS THAT
1969          /WERE SET IN AC AT THE CALL ARE INDICATED IN FLDTAB,
1970          /FRFLD SETS THE FIELD FREE CONDITION IN FLDTAB,
1971          /
1972
1973          /          TAD          SPFLAG /FETCH SPECIAL FLAGS (E.G. CRESBIT) IN AC
1974          /          JMS          GETFLD
1975          /AC B6-11 HOLD VFOLD#. IF NO MORE FREE FLDS ARE AVAILABLE A
1976          /SYSTEM HALT OCCURS,
1977          /USES MON3.
1978
1979          01447  0000  GETFLD, 0
1980          01450  1373          TAD          (2001 /ADD USED AND OCCUPIED CONDITIONS TO
1981          01451  3324          DCA          FLDTM /FLAGS. STORE FLAGS,
1982          01452  1112          TAD          XFLDTAB /SEARCH FLDTAB FOR FREE FLD,
1983          01453  3153          DCA          MON3
1984          01454  1372          TAD          (-VIRMAX-1
1985          01455  3307          DCA          FRFLD /JUST A CTR.
1986          01456  7001  GETFLP, IAC          /SEARCH LOOP,
1987          01457  0553          AND I      MON3 /FREE?
1988          01460  7650          SNA CLA
1989          01461  5266          JMP          GETFLP /YES,
1990          01462  2153          ISZ          MON3
1991          01463  2307          ISZ          FRFLD /INCREMENT COUNT,
1992          01464  5256          JMP          GETFLP
1993          01465  4576          SYSHLT      /FREE FIELDS EXHAUSTED!!!
1994          01466  1324  GETFLF, TAD          FLDTM /GET FLAGS
1995          01467  7040          CMA
1996          01470  0553          AND I      MON3
1997          01471  1324          TAD          FLDTM /ADD TO FLDTAB ENTRY
1998          01472  7500          SMA          /IN CORE?
1999          01473  5303          JMP          GETFL2 /NO,
2000          01474  1120          TAD          C2;FDP=2/YES, SET DATAPRESENT BIT,
2001          0002
2002          01475  3553          DCA I      MON3
2003          01476  1131          TAD C100;FCRES=100 /WILL IT BE CORERESIDENT?
2004          0100
2005          01477  0324          AND          FLDTM
2006          01500  7440          SZA
2007          01501  4227          JMS          VFLOCK /YES, LOCK IT.
2008          01502  5304          JMP          GETFLX
2009          01503  3553  GETFL2, DCA I      MON3
2010          01504  1153  GETFLX, TAD          MON3
2011          01505  0130          AND          C77 /FLDTAB AT BEGIN OF PAGE
2012          01506  5647          JMP          GETFLD
2013
2014          /          TAD          FLDPTR /VFOLD# IN AC B6-11, OTHER BITS DISCARDED,
2015          /          JMS          FRFLD
2016          /USES MON3.
2017
2018          01507  0000  FRFLD, 0          /TEMP FOR GETFLD.

```

2019	01510	0130	AND	C77	
2020	01511	1112	TAD	XFLDTAB	
2021	01512	3153	DCA	MON3	
2022	01513	1553	TAD	MON3	/GET CONTENTS OF FLDTAB.
2023	01514	0131	AND	C100	
2024	01515	7041	CIA		/0 OR -100
2025	01516	7440	SZA		
2026	01517	4227	JMS	VFLOCK	/YES, SO IT WAS LOCKED, UNLOCK IT.
2027	01520	1371	TAD	(7777-FOCCUP-FDP-FCRES-FZREQ-FDATA	
2028	01521	0553	AND	MON3	
2029	01522	3553	DCA	MON3	/RESET CONDITIONS IN FLDTAB.
2030	01523	5707	JMP	FRFLD	
2031	01524	0000		FLDTEM, 0	
2032					
2033	01525	1364		AVPT;(0;AVPT=.-2	
2034	01526	1570			
2035		1525			
2036					
2037	01570	0000			
2038	01571	6274			
2039	01572	7700			
2040	01573	2001			
2041	01574	1303			
2042	01575	1261			
2043	01576	1600			
2044	01577	0500			
2045		1600			PAGE

```

2046      /STORE STUFF OF GENERAL PAGE ZERO INTO FIELD.
2047      /MON3 PTS TO ENTRY OF FIELD IN FLDTAB.
2048      GPZRIN, 0
2049      01600 0000      TAD      C4
2050      01601 1122      AND      MON3
2051      01602 0553      CLL RTR;RAR      /MONINBIT IN LINK.
2052      01603 7112
2053      01604 7010
2054      01605 1553      TAD      MON3
2055      01606 0141      AND      C7773      /MASK OUT POSSIBLE MONINBIT.
2056      01607 1122      TAD      C4      /DONT TOUCH LINK!
2057      01610 3553      DCA      MON3      /SET MONINBIT.
2058      01611 1117      TAD      XCDF70
2059      01612 0553      AND      MON3      /GET CDF TO ACT FLD.
2060      01613 3214      DCA      GPZCDF
2061      01614 6201      GPZCDF, CDF
2062      01615 7430      SZL
2063      01616 5232      JMP      GPZR12      /WAS MONINBIT ALREADY SET?
2064      01617 1377      TAD      (13      /YES; ONLY ADJUST (CIF CDF CUR).
2065      01620 3010      DCA      X0
2066      1634      GPZCT=POCTAD
2067      01621 1376      TAD      (-150+14
2068      01622 3234      DCA      GPZCT
2069      01623 1375      TAD      (TAD 14
2070      01624 3225      DCA      GPZTAD
2071      01625 1000      GPZTAD, TAD
2072      01626 2225      ISZ      GPZTAD
2073      01627 3410      DCA      X0
2074      01630 2234      ISZ      GPZCT
2075      01631 5225      JMP      GPZTAD
2076
2077      /UPDATE GENERAL PAGE 0.
2078      /WHEN THE GENERAL PAGE 0 IS PRESENT IN A FLD, A NUMBER
2079      /OF LOCS OF THAT FLD MUST CONTAIN CIF CDF CUR,
2080      /WHERE CUR IS THE CURRENT ACTUAL FLD.
2081      /THIS ROUTINE FILLS THESE LOCATIONS.
2082      /MON3 POINTS TO THE ENTRY IN FLDTAB OF THE
2083      /FLD TO BE UPDATED.
2084      01632 1374      GPZR12, TAD      (TAD POCLST
2085      01633 3234      DCA      POCTAD
2086      01634 1000      POCTAD, TAD      /GET PTR IN PAGE 0.
2087      01635 7450      SNA
2088      01636 5245      JMP      POCEX      /0 TERMINATES LIST OF LOCS.
2089      01637 3225      DCA      GPZTAD      /TEMP
2090      01640 7326      AC2
2091      01641 1214      TAD      GPZCDF      /CIF CDF CUR IN AC.
2092      01642 3625      DCA      GPZTAD
2093      01643 2234      ISZ      POCTAD
2094      01644 5234      JMP      POCTAD
2095      01645 6201      POCEX, CDF 0
2096      01646 5600      JMP      GPZRIN
2097
2098      01647 0021      POCLST, VVRCDF+1
2099      01650 0026      VVRCDF+1
2100      01651 0052      MYCDF

```

```

2101 01652 0061 VGET+1
2102 01653 0071 VJUMS+1
2103 01654 0076 VPUT+1
2104 01655 0000 0
2105
2106 /***** PAGING ROUTINES
2107 /
2108 /PAGING ROUTINES ARE USED TO DECIDE WHICH VIRTUAL FIELDS ARE IN
2109 /CORE AND WHICH ARE TEMPORARILY STORED ON DISK.
2110 /FROM A VARIETY OF POSSIBLE PAGING ALGORITHMS WE CHOOSE A
2111 /VERY SIMPLE ONE: WE SWAP OUT THE FIELD THAT WAS LONGEST IN CORE.
2112 /THIS ALGORITHM WAS CHOSEN BECAUSE IT IS EASY TO PROGRAM (SEE
2113 /VFLESS), DOESNOT TAKE ANY TIME AS LONG AS NO FIELDSWAPS ARE
2114 /NEEDED AND DOESNOT WORK TOO BAD.
2115 /BETTER ALGORITHMS MIGHT TAKE ADVANTAGE OF THE FACT THAT WE KNOW
2116 /WHICH FIELDS ARE LAST USED, TO DO SO ONE MUST REMOVE THE SLASHES
2117 / IN FCHECK AND WRITE SOME ROUTINE VFSIGN. FCHECK IS CALLED
2118 /AT LEAST TWICE FOR EACH TASK SWITCH!
2119 /THE PAGING ROUTINES CONTROL THE CONTENTS OF THE ARRAY:
2120 /CORMAP[0:ACTMAX], ONE ENTRY FOR EACH ACTUAL FIELD.
2121 /LAYOUT OF THE ENTRIES IN CORMAP:
2122 /      B0-5: COUNTS HOW MANY TIMES THIS FIELD IS LOCKED IN
2123 /            CORE (E.G., WHEN DATABREAK PROCESSES USE THIS
2124 /            FIELD).
2125 /      B6-11: THE VIRTUAL FIELD THAT RESIDES IN THIS FIELD,
2126
2127 /VFLESS.
2128 /ROUTINE TO DETERMINE WHICH VFLO IS TO BE SWAPPED OUT,
2129 /WE TAKE THE ONE THAT WAS LONGEST IN CORE, UNLESS IT WAS LOCKED.
2130 /THEN WE TAKE THE ONE THAT WAS NEXT LONGEST IN CORE ETC....
2131 /IF ALL FIELDS ARE LOCKED, THE ERROR RETURN IS TAKEN.
2132 /      JMS      VFLESS
2133 /      JMP      ERROR      /AC=0, ALL FLDS ARE LOCKED.
2134 /      ...      /AC=PTR IN CORMAP.
2135
2136 01656 4722 VFNEXT, CORMAP /POINTER TO (FLD THAT WAS LONGEST IN CORE)-1,
2137
2138 01657 0000 VFLESS, 0
2139 01660 1373 TAD      (-ACTMAX-1
2140 01661 3200 DCA      VFLCT /COUNT THE NUMBER OF FLDS WE TRIED.
2141 01662 1256 VFLTRY, TAD      VFNEXT /KEEP THIS POINTER CIRCULING IN CORMAP.
2142 01663 1372 TAD      (-CORMAP-ACTMAX
2143 01664 7640 SZA CLA
2144 01665 5270 JMP      .+3
2145 01666 1170 TAD      ZCORMAP
2146 01667 3256 DCA      VFNEXT
2147 01670 2256 ISZ      VFNEXT
2148 01671 1656 TAD      VFNEXT /IS THE CANDIDATE FIELD LOCKED?
2149 01672 0145 AND      C7700
2150 01673 7650 SNA CLA
2151 01674 5300 JMP      VFLF /NO, FIELD TO BE SWAPPED OUT IS FOUND
2152 01675 2200 ISZ      VFLCT /ALL FLDS CHECKED?
2153 01676 5262 JMP      VFLTRY /NO,
2154 01677 5657 JMP      VFLESS /YES, ALL LOCKED. TAKE ERROR RETURN.
2155 01700 1256 VFLF, TAD      VFNEXT /RETURN PTR IN CORMAP,

```

```
2156 01701 2257      ISZ      VFLESS
2157 01702 5657      JMP      VFLESS
2158
2159      1600 VFLCT=GPZRIN
2160 01703 1525      AVPT;(0;AVPT=-2
2161 01704 1771
2162      1703
2163 01771 0000
2164 01772 3047
2165 01773 7770
2166 01774 1247
2167 01775 1014
2168 01776 7644
2169 01777 0013
2170      2000 PAGE
```

```
2171 /APP. A3. MC8.3. MONITOR TASKS.
2172 /***** MONITOR TASKS *****/
2173 /
2174 /TASKS IN MC8 MAY BE DIVIDED INTO 3 GROUPS:
2175 /MONITOR TASKS THESE TASKS ARE EFFECTIVELY A PART OF THE
2176 / MONITOR. THEY ARE BUILT AS A TASK JUST BECAUSE IT
2177 / IS CONVENIENT TO THINK OF THEM AS TASKS,
2178 / SOMETIMES THEY ARE ACTIVATED VIA THE SCHEDULER,
2179 / SOMETIMES THE MONITOR DIRECTLY JUMPS INTO SUCH
2180 / TASKS.
2181 / NOTE: NO OTHER TASKS (NOT EVEN MONITOR
2182 / TASKS) CAN INTERRUPT ACTIVE MONITOR TASKS.
2183 /SYSTEM TASKS THESE TASKS CONSTITUTE THE MULTIPROGRAMMING MODE
2184 / OF MC8. THEY ARE WRITTEN AND ASSEMBLED AS
2185 / SEPERATE MODULES, KNOWING LITTLE ABOUT THE
2186 / MONITOR AND IN PRINCIPLE NOTHING ABOUT OTHER
2187 / TASKS. OF COURSE ONE COULD WRITE FAMILIES OF
2188 / TASKS USING WELL DEFINED INTERFACES.
2189 /PROTECT TASKS THESE TASKS CONSTITUTE THE MC8 TIME SHARING MODE.
2190 / TO EACH TIME SHARING USER A BARE MACHINE HAVING
2191 / AN APPROPRIATE CONFIGURATION IS SIMULATED. OTHER
2192 / PARTS OF THE SYSTEM ARE PROTECTED AGAINST ANY
2193 / OBSCURE PROGRAMS EXECUTED BY THESE TASKS. THEY
2194 / HAVE NO DIRECT ACCESS TO ANY PART OF THE SYSTEM
2195 / AND RUN WITH USERMODE ON.
2196 /
2197 /THIS SECTION CONTAINS THE FOLLOWING MONITOR TASKS:
2198 /MTSWAP.
2199 / THIS TASK SWAPS VIRTUAL FIELDS INTO AND OUT OF ACTUAL
2200 / CORE. IT UPDATES THE ARRAYS FIELDTABLE AND CORMAP.
2201 / IT IS ENTERED FROM THE SCHEDULER (UPON SWAP COMPLETION).
2202 /MTSDSK.
2203 / SYSTEM DISK MANAGER. THIS TASK SWAPS INFORMATION FROM AND
2204 / TO THE SYSTEM DISK.
2205 / IT ACCEPTS MESSAGES FROM ANY MONITOR AND SYSTEM TASK,
2206 / THE MESSAGE SWAPMS (COMMUNICATION MESSAGE WITH MTSWAP) IS
2207 / GIVEN SPECIAL TREATMENT.
2208 / IT IS ENTERED ONLY FROM THE SCHEDULER.
2209 /MTTIME.
2210 / THIS TASK KEEPS TRACK OF THE SYSTEM TIME (DOUBLE
2211 / PRECISION COUNTER IN UNITS OF 0.1 SEC). ON START
2212 / SYSTEM TIME IS SET TO 0.
2213 / MOREOVER IT HANDLES THE TIME OUT QUEUE. TASKS THAT WANT
2214 / TO BE CONTINUED AFTER SOME SECONDS CAN SPECIFY A TIME OUT
2215 / REQUEST TO THE MONITOR. THEIR TIME OUT VALUE MUST BE
2216 / STORED IN AC. THESE TASKS ARE ENTERED IN THE TIME OUT
2217 / QUEUE AND EACH 0.1 SEC THEIR TIME OUT VALUE IS
2218 / INCREMENTED. ON OVERFLOW THEY ARE REINSERTED IN THE
2219 / RUNQ.
2220 /MTFTCH.
2221 / A RUNNABLE TSK THAT IS NOT IN CORE IS ALLOCATED IN CORE,
2222 / SWAPPED IN AND INSERTED IN THE RUNQ.
```

```

2223      /***** FCHECK AND MTSWAP.
2224      /
2225      /FCHECK (ASSERT FIELD IN CORE).
2226      /CALLED WITH VFLD# IN AC.
2227      /IF VFLD IN CORE, RETURN CDF TO ACTFLD IN AC.
2228      /ELSE IF NO SWAP IS PENDING YET (SWAPMS[2]#0) ISSUE A SWAP
2229      /    REQUEST. SAFE THE CURRENT TASK (IF ANY) WITH THE REQUESTED
2230      /    VFLD AS DF.
2231      /    RETURN TO THE SCHEDULER.
2232      /
2233      /EACH TASK CAN ACCESS TWO VFLDS: IF AND DF. WHEN SCHEDULING A
2234      /TASK THE SCHEDULER CALLS FCHECK TO ASSERT THAT BOTH THESE
2235      /FIELDS ARE IN ACTUAL CORE. IF THEY ARE NOT FCHECK INITIATES A
2236      /SCHEDULE OF THE NEXT IMPORTANT TASK (JMP SCEDN1).
2237      /
2238      /IF A TASK WANTS TO CHANGE ITS DF (NOTE: CHANGING IF IS NOT
2239      /ALLOWED) IT CALLS FCHECK TO ASSERT THAT THE NEW DF IS IN ACTUAL
2240      /CORE. IF IT IS NOT, FCHECK JUMPS INTO MONSAF, SAFING THE
2241      /CURRENT TASK WITH THE REQUESTED NEW DF.
2242
2243      02000  0000  FCHECK, 0          /AC =VFLD#
2244      02001  0130          AND      C77
2245      02002  1112          TAD      XFLDTAB
2246      02003  3153          DCA      MON3
2247      02004  1553          TAD I    MON3
2248
2249          /          SMA          /AC:=CONTENTS OF FLDTAB.
2250          /          JMP      FCSWAP /IN CORE?
2251          /          AND      C70
2252          /          CLL RTR
2253          /          RAR
2254          /          JMS      VFSIGN /SIGNAL TO VIRTUAL CORE MANAGER WHICH
2255          /          /FLDS ARE USED (USED IN PAGING
2256          /          /ALGORITHM).
2257          /          TAD I    MON3
2258          /          AND      XCDF70 /YES; COMPUTE CDF TO ACT FLD.
2259          /          JMP I    FCHECK
2260      02005  0117          AND      XCDF70
2261      02006  7510          SPA
2262      02007  5600          JMP I    FCHECK
2263
2264      02010  7200  FCSWAP, CLA          /WE HAD RUBBISH IN AC.
2265      02011  1237          TAD      SWAPMS /SWAP REQUEST PENDING?
2266      02012  7650          SNA CLA /IF NOT WHICH FIELD SHALL WE SWAP OUT?
2267      02013  4777          JMS      VFLESS
2268      02014  5776          JMP      FCHEX /NONE! ALL LOCKED. DANGER FOR DEAD LOCK!!
2269      02015  3237          DCA      SWAPMS /PTR IN CORMAP.
2270      02016  1637          TAD I    SWAPMS /GET VFLD# OF SWAP OUT FIELD
2271      02017  1112          TAD      XFLDTAB
2272      02020  3240          DCA      SWAPMS+1/SWAP OUT-ENTRY IN FIELDTABLE.
2273      02021  7350          AC3777
2274      02022  0640          AND I    SWAPMS+1
2275      02023  3640          DCA I    SWAPMS+1/CLEAR INCORE BIT.
2276      02024  1153          TAD      MON3
2277      02025  3241          DCA      SWAPMS+2/SWAP IN-ENTRY IN FLDTAB.

```

2278	02026	6002	IOF	/\
2279	02027	1375	TAD	(SWAPMS-2/\PTR TO MS LINKWORD
2280	02030	3154	DCA	INT0 /\
2281	02031	1374	TAD	(MTSDSK-1/\
2282	02032	4571	JMS	ZSNDREP /\SEND THIS MS TO MTSDSK.
2283				/IT WILL BE REPORTED TO MTSWAP.
2284	02033	6001	ION	/\
2285	02034	5776	JMP	FCHEX /
2286				
2287			/SWAP MESSAGE.	
2288	02035	5035	MTSWAP	/SENDER WORD
2289	02036	0000	0	/LINK WORD
2290	02037	0000	SWAPMS, 0	/PTR TO COREMAP[ACTFLD]
2291	02040	0000	0	/PTR TO FLDTAB[SWAP OUT FLD]
2292	02041	0000	0	/PTR TO FLDTAB[SWAP IN FLD]
2293				

```

2294      /MTSWAP.
2295      /THIS TASK IS ACTIVATED AFTER A SWAP OF VFlds,
2296      /IT UPDATES FLDTAB AND CORMAP. MOREOVER ITS SCHEDULE GUARANTEES
2297      /A COMPLETE REEXAMINATION OF THE RUNQ'S, GENERALLY THE TASK IN
2298      /CHARGE OF WHICH THE SWAP WAS ISSUED WILL PROFIT FROM THAT
2299      /REEXAMINATION. THE SWAPMS IS ALWAYS SENT TO MTSDBK BY FCHECK AND
2300      /REPORTED TO MTSWAP.
2301
2302      VFSWLP,
2303      /      CALL      /PERPETUAL LOOP,
2304      /      WTRP
2305      /      SWAPMS
2306      02042  7200  MTSWIN, CLA      /PTR TO SWAPMS WAS IN AC.
2307      /      TAD      SWAPMS
2308      /      TAD      (=CORMAP
2309      /      JMS      VFSIGN /SIGNAL TO VIRT.CORE MANAGER WHICH FLDS
2310      02043  1641  TAD I      SWAPMS+2/ARE USED. GET STATUS OF SWAP IN FLD FROM
2311      02044  0373  AND      (7777-4270-FDP /FLDTAB,
2312      02045  1350  TAD      SWNODE /ACTFLD STILL IN SWNODE B6-8,
2313      02046  1372  TAD      (4200+FDP/SET UP IN CORE +DATA PRESENT,
2314      02047  3641  DCA I      SWAPMS+2
2315      02050  1131  TAD      C100;FCRES=100 /SWAP IN FIELD CORERESIDENT?
2316      0100
2317      02051  0641  AND I      SWAPMS+2
2318      02052  1241  TAD      SWAPMS+2/PERHAPS AC=100, ONE LOCK,
2319      02053  0132  AND      C177 /NOTE: FLDTAB AT BEGIN OF PAGE;
2320      02054  3637  DCA I      SWAPMS /SET NEW VFld IN COREMAP,
2321      02055  1241  TAD      SWAPMS+2
2322      02056  3153  DCA      MON3
2323      02057  1641  TAD I      SWAPMS+2      /GET STATUS OF SWAP IN FLD.
2324      02060  0173  AND      Z1000;FDATA=1000 /NO NEED FOR UPDATING PAGE 0
2325      1000
2326      02061  7650  SNA CLA      /IF IT IS GIVEN OUT FOR DATA,
2327      02062  4771  JMS      GPZRIN /ADJUST GENERAL PAGE 0.
2328      02063  3237  MTSWEX, DCA      SWAPMS /SIGNAL FREE FOR NEW SWAP.
2329      /      JMP      VFSWLP
2330
2331      /SET MONTSK TO WAIT FOR REPORT WITHOUT SAVING. THIS SPARES TIME.
2332      02064  1370  TAD      (=SWAPMS+2
2333      02065  3767  DCA      MTSWAP-3
2334      02066  5766  JMP      WTRPMT

```

```

2335      /MTSDSK SYSTEM DISK MANAGER.
2336      /SWAP TO AND FROM THE SYSTEM DISK.
2337      /THE SYSTEM DISK HOLDS:
2338      /      VFLDSLOTS; ROOM ON BACKGROUND STORAGE FOR EACH VFLD,
2339      /      TASK LIBRARY; SYSTEM TASKS THAT MAY BE RUN.
2340      /      OTHER; ANY ROOM LEFT ON THE SYSTEM DISK.
2341      /THE SYSTEM DISK IS DIVIDED INTO UPTO 8 LOGICAL UNITS, THAT
2342      /EACH CONTAIN UPTO 4096 BLOCKS OF 256 WORDS,
2343      /A VFLDSLOT OCCUPIES 16 BLOCKS.
2344      /THE VFLDSLOTS ARE ALLOCATED FROM VFBLOK (MULTIPLE OF 200CT)
2345      /UPWARD. THE TASK LIBRARY IS ALLOCATED FROM TSBLOK UPWARD.
2346      /BOTH THE VFLDSLOTS AND THE TASK LIBRARY ARE ASSUMED TO RESIDE ON
2347      /LOGICAL UNIT 0.
2348      /
2349      /MTSDSK OPERATES IN TWO MODES: NORMAL MODE AND FIELD SWAP MODE.
2350      /EACH TIME A NEW MESSAGE IS ACCEPTED (BY CALL;WTMS) FIRST THE
2351      /MODE IS SELECTED. FIELD SWAP MODE IS ENTERED IF THE ACCEPTED
2352      /MESSAGE IS THE SWAPMS. OTHERWISE NORMAL MODE IS ENTERED.
2353      /
2354      /NORMAL MODE.
2355      /MESSAGE LAYOUT:
2356      /W0:      R/W, NNNNN, FFF, UUU
2357      /      W/R=1: WRITE ON, W/R=0: READ FROM DISK.
2358      /      NNNNN: 5-BIT NUMBER OF PAGES, 0=32 PAGES.
2359      /      NOTE: 1 BLOCK =2 PAGES.
2360      /      FFF:      3-BIT ACTFLD# OF TRANSFER.
2361      /      UUU:      3-BIT LOGICAL UNIT NUMBER.
2362      /W1:      CORE ADDRESS.
2363      /W2:      BLOCKNUMBER.
2364      /NOTE:!!!! THE CALLING TASK MUST ASURE THAT THE ACTUAL
2365      /      FIELD IN WHICH THE TRANSFER TAKES PLACE IS IN
2366      /      CORE. THIS IS MOST EASILY DONE BY USING
2367      /      THE DFPARM OPTION WHILE SENDING THE MS.
2368      /      THIS OPTION COPIES THE DATAFIELD OF THE TASK
2369      /      INTO BIT 6-8 OF THE FIRST INFO WORD OF THE MS.
2370      /THE FIELD CANNOT BE SWAPPED OUT MEANWHILE, AS POSSIBLE MSS TO
2371      /SWAP OUT THAT FLD, WILL BE AFTER THE CURRENT ONE IN THE RCQ.
2372      /
2373      /FIELD SWAP MODE.
2374      /MESSAGE LAYOUT:
2375      /W0:      ACTFLD# +CORMAP
2376      /W1:      SWAP OUT VFLD# +FLDTAB
2377      /W2:      SWAP IN VFLD# +FLDTAB.
2378      /IN THIS MODE FIRST THE SWAP OUT FIELD IS WRITTEN ON DISK IN ITS
2379      /VFLDSLOT. SUBSEQUENT THE SWAP IN FIELD IS READ IN CORE FROM
2380      /ITS VFLDSLOT.
2381      /OPTIMIZATION OCCURS IF IN ONE OF THESE FLDS NO DATA ARE PRESENT.
2382
2383      SDNORM,
2384      IFDEF STATX <
2385      02067 2353      ISZ      SDNCT
2386      02070 5273      JMP      .+3
2387      02071 2354      ISZ      SDNCT+1
2388      02072 7000      NOP
2389      >

```

```

2390 02073 1300 TAD SD MSP /ENTER NORMAL MODE.
2391 02074 4765 JMS SDGO /PASS MESSAGE TO SDGO ROUTINE,
2392 /RETURNS COMPLETION STATUS IN AC.
2393 02075 3700 DCA SD MSP /COMPLETION STATUS TO MS.
2394 02076 4054 SDEX, CALL /REPORT MESSAGE; BOTH MODES.
2395 02077 0013 RP
2396 02100 0000 SD MSP, 0
2397 02101 4054 SD IN, CALL /WAIT FOR MESSAGE
2398 02102 0014 WTMS
2399 02103 3300 DCA SD MSP /STORE PTR TO MESSAGE
2400 02104 1300 TAD SD MSP /SELECT MODE, NORMAL OR FIELD SWAP.
2401 02105 1364 TAD (-SWAPMS
2402 02106 7640 SZA CLA
2403 02107 5267 JMP SD NORM /NORMAL.
2404 IFDEF STATX <
2405 02110 2355 ISZ SDFCT
2406 02111 5314 JMP .+3
2407 02112 2356 ISZ SDFCT+1
2408 02113 7000 NOP
2409 >
2410 02114 7330 AC4000 /FIELD SWAP MODE,
2411 /COMPUTE MESSAGE FOR SDGO ROUTINE,
2412 02115 1640 TAD SWAPMS+1/GET ACTFLD# OF SWAPFLD.
2413 02116 0363 AND (4070 /AC=40X0, WRITE FLD X ON DISK 0.
2414 02117 3350 DCA SWNODE
2415 02120 1240 TAD SWAPMS+1
2416 02121 4330 JMS SDSWP /JUST A LITTLE ROUTINE TO SAFE CODE.
2417 02122 7350 SDSW1, AC3777
2418 02123 0350 AND SWNODE /SET TO READ
2419 02124 3350 DCA SWNODE
2420 02125 1241 TAD SWAPMS+2/SWAP IN FIELD.
2421 02126 4330 JMS SDSWP
2422 02127 5276 JMP SDEX
2423
2424 02130 0000 SDSWP, 0 /COMPUTE MESSAGE IN SWNODE AND CALL SDGO.
2425 02131 3352 DCA SWNODE+2/TEMP
2426 02132 7326 AC2;FDP=2
2427 0002
2428 02133 0752 AND SWNODE+2/DATA PRESENT IN FLD TO BE SWAPPED?
2429 02134 7650 SNA CLA
2430 02135 5730 JMP SDSWP /NO, NO SWAP NEEDED.
2431 02136 1352 TAD SWNODE+2
2432 02137 1362 TAD (VFBLOK/20-FLDTAB/GET VFLD# +(VFBLOK/20)
2433 02140 7106 CLL RTL;RTL /*20. COMPUTE BLOCKNUMBER
2434 02141 7006
2435 02142 3352 DCA SWNODE+2
2436 02143 1361 TAD (SWNODE /PASS PTR TO MS TO SDGO ROUTINE,
2437 02144 4765 JMS SDGO /RETURNS COMPLETIONSTATUS IN AC,
2438 02145 7440 SZA /ANY ERROR?
2439 02146 4576 SYSLT /THAT'S THE END OF ALL!
2440 02147 5730 JMP SDSWP
2441
2442 02150 0000 SWNODE, 0;0;0 /3TEMPS FOR MS.
2443 02151 0000
2444 02152 0000

```

```
2445
2446 02153 0000 IFDEF STATX <SDNCT, 0;0 >
2447 02154 0000
2448 02155 0000 IFDEF STATX < SDFCT, 0;0 >
2449 02156 0000
2450
2451 IFNDEF STATX <AVPT;(0;AVPT=-2 >
2452
2453 02161 2150
2454 02162 3400
2455 02163 4070
2456 02164 5741
2457 02165 2202
2458 02166 2644
2459 02167 5032
2460 02170 5743
2461 02171 1600
2462 02172 4202
2463 02173 3505
2464 02174 5102
2465 02175 2035
2466 02176 0744
2467 02177 1657
2468 2200 PAGE
```

```

2469 02200 0000 SDPTR, 0 /MUST BE AT BEGIN OF PAGE,
2470 02201 0000 IFDEF STATX <SDERCT, 0 /ERROR COUNT >
2471
2472 /SDGO ROUTINE.
2473 /CALLED WITH PTR TO MS (NORMAL MODE MESSAGE LAYOUT) IN AC.
2474 /EXECUTE SWAP.
2475 /REPORT STATUS AFTER TRANSFER IN AC; 0: OK, OTHER: SOME ERROR,
2476
2477 IFDEF RK8E <
2478 IFZERO SYDISK-RK8E <
2479 02202 0000 SDGO, 0
2480 02203 3200 DCA SDPTR /PTR TO MS.
2481 02204 7346 ACM3
2482 02205 3321 DCA RKERCT /RKSYS! 3 TIMES TRY OVER.
2483 02206 4054 CALL /START UP THE INTERRUPTDRIVEN SECTION TO
2484 02207 0027 SIMINTR /EXECUTE THE SWAP.
2485 02210 2240 RKRTY /START UP AT RKRTY,
2486 02211 4054 CALL
2487 02212 0005 WTRP
2488 02213 2325 RKMS, RKMS
2489 02214 7200 CLA /SWAP COMPLETED, REMOVE DIRT FROM AC,
2490 02215 1200 TAD SDPTR /GET COMPLETIONSTATUS IN AC,
2491 02216 5602 JMP SDGO
2492
2493 /DRIVEN BY RK8E INTERRUPTS.
2494 /RK8E IS CONNECTED TO SYWAIT+2.
2495 /DEAF ROUTINE.
2496
2497 02217 0377 RKER2, AND (1002 /!TREAT ERROR,
2498 02220 7650 SNA CLA /!
2499 02221 5235 JMP RKER3 /!NO NEED FOR RECALLIBRATION,
2500 02222 7326 AC2 /!RECALLIBRATE,
2501 02223 6742 DCLR /!
2502 02224 4330 JMS SYWAIT /!WAIT FOR COMMAND ACCEPT,
2503 02225 7327 AC6 /!
2504 02226 0322 AND RKCMD /!GET DRIVE#.
2505 02227 1245 TAD RK600 /!ADD INTR WHEN SEEK DONE.
2506 02230 6746 DLDC /!
2507 02231 4330 JMS SYWAIT /!WAIT FOR RECALLIBRATE COMPLETE.
2508 02232 1333 TAD RKINTR /!GET STATUS
2509 02233 7440 SZA /!ANY ERROR BITS IN STATUS?
2510 02234 5312 JMP RKERR /!YES.
2511 02235 7344 RKER3, ACM2 /!RESET PTR FOR RETRY,
2512 02236 1200 TAD SDPTR /!
2513 02237 3200 DCA SDPTR /!
2514 /FALL INTO RETRY.
2515
2516 02240 1600 RKRTY, TAD SDPTR /!GET R/W +FFF+UUU
2517 02241 0376 AND (4077 /!
2518 02242 1375 TAD (400 /!ENABLE INTERRUPT WHEN DONE,
2519 02243 3322 DCA RKCMD /!
2520 02244 1145 TAD C7700 /!
2521 02245 0600 RK600, AND SDPTR /!GET NNNNN
2522 02246 2200 ISZ SDPTR /!
2523 02247 7045 CIA RAL /!SET RK PAGE COUNT,

```

```

2524 02250 7130      STL RAR      /!AC MIGHT BE =0.
2525 02251 3326      DCA      RKPAG  /!
2526 02252 1600      TAD      SDPTR  /!
2527 02253 2200      ISZ      SDPTR  /!
2528 02254 6744      DLCA      /!LOAD CORE ADDRESS REGISTER.
2529 02255 1600      TAD      SDPTR  /!
2530 02256 3327      DCA      RKBLK  /!BLOCK NUMBER.
2531 02257 3325      DCA      RKCHEK /!FORCE CHECK HEADER FIRST TIME.
2532 02260 7100      RKLP,    CLL      /!
2533 02261 1326      TAD      RKPAG  /!
2534 02262 7450      SNA      /!READY?
2535 02263 5315      JMP      RKDONE /!
2536 02264 1133      TAD      C200   /!UPDATE THIS COUNTER
2537 02265 7420      SNL      /!OVERFLOW?
2538 02266 3326      DCA      RKPAG  /!NO. STORE NEW VALUE.
2539                      /!IF OVERFLOW, AC=0 OR =100.
2540                      /!100 MEANS ONE PAGE ONLY.
2541 02267 1325      TAD      RKCHEK /!CHECK HEADER BIT?
2542 02270 1322      TAD      RKCMD  /!GET COMMAND
2543 02271 6746      DLDC      /!
2544 02272 1327      TAD      RKBLK  /!
2545 02273 6743      DLAG      /!
2546 02274 7430      SZL      /!
2547 02275 3326      DCA      RKPAG  /!IF OVERFLOW, CLEAR RKPAG TO SIGNAL END
2548 02276 2327      ISZ      RKBLK  /!OF TRANSFER, INCREMENT BLOKNUM
2549 02277 7410      SKP      /!
2550 02300 2322      ISZ      RKCMD  /!NOTE: ONLY EVEN LOGICAL UNITS CAN
2551                      /!OVERFLOW. IN THAT CASE WE CONTINUE
2552                      /!TRANSFERRING TO THE NEXT LOGICAL UNIT.
2553 02301 1327      TAD      RKBLK  /!TEST FOR CHECK HEADER CONDITION.
2554 02302 0125      AND      C37
2555 02303 7640      SZA      CLA      /!
2556 02304 1374      TAD      (1000 /!NO CHECK HEADER NEXT TIME.
2557 02305 3325      DCA      RKCHEK /!
2558 02306 4330      JMS      SYWAIT /!WAIT FOR FLAG.
2559 02307 1333      TAD      RKINTR /!GET STATUS AGAIN
2560 02310 7450      SNA      /!RETURNS WITH DRST;RAL IN AC. 0 IF
2561                      /!NO ERROR OCCURRED.
2562 02311 5260      JMP      RKLP    /!
2563
2564                      RKERR,      /!
2565 02312 2201      IFDEF STATX <ISZ SDERCT;/NOP> /!
2566 02313 2321      ISZ      RKERCT /!INCREMENT ERROR COUNT.
2567 02314 5217      JMP      RKER2  /!TRY ONCE MORE
2568 02315 3200      RKDONE, DCA      SDPTR  /!SAFE STATUS.
2569 02316 1213      TAD      RKMSP  /!REPORT MS.
2570 02317 4330      JMS      SYWAIT /!
2571 02320 5317      JMP      .-1    /!IGNORE OBSCURE INTERRUPTS.
2572
2573 02321 0000      RKERCT, 0      /3 ERROR COUNT.
2574 02322 0000      RKCMD, 0      /RK COMMAND.
2575
2576 02323 5103      MTSDSK      /SENDERWORD
2577 02324 0000      0          /LINK WORD.
2578
2578                      RKMS,

```

```
2579 02325 0000 RKCHK, 0
2580 02326 0000 RKPAG, 0
2581 02327 0000 RKBLK, 0
2582
2583 02330 0000 SYWAIT, 0          /!WAIT HERE UNTIL NEW INTERRUPT OCCURS,
2584 02331 3333          DCA      RKINTR /!POSSIBLE MS TO BE REPORTED,
2585 02332 4515          IEXIT      /!
2586 02333 0000 RKINTR, 0          /!
2587 02334 6745          DRST      /!READ STATUS
2588 02335 7104          CLL RAL    /!
2589 02336 3333          DCA      RKINTR /!SAVE TATUS
2590 02337 6742          DCLR      /!CLEAR FLAGS
2591 02340 5730          JMP I     SYWAIT /!
2592 >>
```

```

2593      IFDEF RF08 <
2594      IFZERO SYDISK-RF08 <
2595
2596          RFWC=7750; RFCA=7751
2597          DXAL=6643
2598          DIML=6615
2599          DIMA=6616
2600          DCMA=6601
2601
2602      SDGO, 0
2603          DCA      SDPTR  /PTR TO MS.
2604          TAD I    SDPTR  /FETCH UNIT NUMBER.
2605          AND      C7
2606          SNA
2607          JMP      .+3
2608          AC2      /SET NXTENT UNIT CONDITION.
2609          JMP I    SDGO
2610          TAD I    SDPTR  /GET FIELD
2611          AND      C70
2612          TAD      RF500  /ENABLE ERROR AND COMPLETION.
2613          DIML     /LOAD STATUS
2614          TAD I    SDPTR  / R/W
2615          SPA CLA
2616          AC2
2617          TAD      (6603
2618          DCA      RFINS  /6603 IF READ, 6605 IF WRITE.
2619          TAD I    SDPTR
2620          ISZ      SDPTR
2621          AND      C7700  /NUMBER OF PAGES IN B1-5
2622          CLL RAL  /NUMBER OF WORDS
2623          CIA
2624          DCA      RFWC  /WORD COUNT.
2625          CMA
2626          TAD I    SDPTR  /CORE ADDRESS.
2627          ISZ      SDPTR
2628          DCA      RFCA
2629          TAD I    SDPTR  /BLOCK # ON DISK
2630          CLL RTR;RTR
2631          DCA      RFTEMP
2632          TAD      RFTEMP
2633          DXAL     /LOAD DMA
2634          TAD      RFTEMP
2635          RAR
2636          AND      C7600
2637      RFINS, HLT      /GO!
2638          CALL
2639          WTRP
2640          RFMS
2641          CLA
2642          TAD      RFSTAT
2643          JMP I    SDGO  /RETURNS STATUS IN AC.
2644
2645      RFINTR, 0      /! INTERRUPT ARRIVES HERE!
2646          DIML     /! CLEAR INTR ENABLE.
2647          DIMA     /! READ STUTUS

```

```
2648          AND      C7      /!
2649          IFDEF STATX <
2650              SZA      /!
2651              ISZ      SDERCT /!
2652              NOP      /!
2653      >
2654              DCA      RFSTAT /!
2655              DXAL      /!CLEAR POSSIBLE NXTENT CONDITION.
2656              DCMA      /!CLEAR FLAGS
2657              IEXIT      /!
2658              RFMS      /!
2659
2660              MTSDSK      /SENDER WORD
2661              0            /LINK WORD
2662      RFMS,
2663      RFTEMP, 0
2664      RFSTAT, 0
2665      RF500, 500
2666      >>
```

```

2667      /IDLE.
2668      /THE IDLELOOP.
2669      /THIS IS ESSENTIALLY NOT A MONITORTASK,
2670      /IN FACT WE START BY CLEARING CURTSK, THUS ENABLING ANY OTHER
2671      /TASK TO INTERRUPT US WITHOUT LOOSING TIME FOR SAFING,
2672
2673      IFDEF STATX <
2674      02341 1447 IDNSHL, -TCKLEN
2675      02342 0000 IDNS, 0;0
2676      02343 0000
2677      02344 1447 IDSWHL, -TCKLEN
2678      02345 0000 IDSW, 0;0
2679      02346 0000
2680      02347 0000 IDPTR, 0
2681
2682      02350 6001 IDLP, ION /!
2683      02351 3564 DCA I CURTSK /SPEND SOME TIME AND CLEAR AC, INNOCENT.
2684      >
2685      02352 3164 IDLEIN, DCA CURTSK
2686      IFNDEF STATX < IDLP, JMP .>
2687      IFDEF STATX < /COUNT IDLE TIME
2688      02353 1773 TAD I (SWAPMS /SWAPPING?
2689      02354 7650 SNA CLA
2690      02355 1372 TAD (IDNSHL-1/NO.
2691      02356 7450 SNA
2692      02357 1371 TAD (IDSWHL-1/YES.
2693      02360 3347 DCA IDPTR /PTR TO COUNT VARIABLES.
2694      02361 1370 TAD (-TCKLEN
2695      02362 6002 IOF /! DONT INTERRUPT OVERFLOW CONDITION
2696      02363 2347 IDLP2, ISZ IDPTR /!
2697      02364 2747 ISZ I IDPTR /!
2698      02365 5350 JMP IDLP /!
2699      02366 3747 DCA I IDPTR /!
2700      02367 5363 JMP IDLP2 /!
2701      >
2702
2703      IFNDEF RK8E <AVPT;(0;AVPT=-2>
2704      IFDEF RK8E <
2705      IFNZRO SYDISK-RK8E <AVPT;(0;AVPT=-2>
2706      >
2707
2708
2709      02370 1447
2710      02371 2343
2711      02372 2340
2712      02373 2037
2713      02374 1000
2714      02375 0400
2715      02376 4077
2716      02377 1002
2717      2400 PAGE

```

```

2718
2719 /MTFCH;
2720 02400 4054 MTFTIN, CALL
2721 02401 0014 WTMS
2722 02402 4014 MSFREE /FREE MS, GET CONTENTS OF W0.
2723 02403 3107 DCA BASE /PTR TO TCB[5].
2724 02404 4060 GET; 1 /GET STSK#
2725 02405 0001
2726 02406 0132 AND C177
2727 02407 7124 STL RAL
2728 02410 1116 TAD ZSTL /
2729 02411 3151 DCA MON1 /PTR TO STATIC TASK LIST[TSK,1]
2730 02412 1551 TAD MON1 /GET LENGTH +FCRES +FZREQ
2731 02413 0377 AND (537
2732 02414 3161 DCA MARG1 /ARG FOR PGRQN.
2733 02415 1161 TAD MARG1 /BUILD MS FOR MTSDSK IN FCHMS.
2734 02416 0125 AND C37
2735 02417 7002 BSWR
2736 IFNDEF PDP8E <RAR>
2737 02420 3334 DCA FTCHMS /STORE LENGTH AND LOGICAL UNIT 0.
2738 02421 1161 TAD MARG1 /SET UP COUNTER FOR RELOCATION LOOP.
2739 02422 0125 AND C37
2740 02423 7041 CIA
2741 02424 3331 DCA MTFCT
2742 02425 4776 JMS PGRQN /REQUEST FOR APPROPRIATE SPACE
2743 02426 3335 DCA MTFPAG /TEMP.
2744 02427 1335 TAD MTFPAG /GET FLD BITS.
2745 02430 0130 AND C77
2746 02431 3255 DCA MTFFLD
2747 02432 1335 TAD MTFPAG /PAGE BITS
2748 02433 0146 AND C7600
2749 02434 3335 DCA MTFPAG /ADDRESS OF FIRST PAGE.
2750 02435 2110 ISZ X /PTR TO TCB[7] FLDS.
2751 02436 1255 TAD MTFFLD
2752 02437 7002 BSWR
2753 IFNDEF PDP8E <RAR>
2754 02440 1255 TAD MTFFLD
2755 02441 3510 DCA X /IF AND DF TO FLD.
2756 02442 4060 GET; 11 /FETCH TCB[14]
2757 02443 0011
2758 02444 0132 AND C177
2759 02445 1335 TAD MTFPAG
2760 02446 3510 DCA X /UPDATE TCB[14]
2761 02447 2110 ISZ X /PTR TO TCB[15] DISK ADDRESS
2762 02450 7350 AC3777 /11-BIT BLOCKNUMBER.
2763 02451 0510 AND X /GET DISK ADDRESS
2764 02452 1375 TAD (TSBLOK /ADD STARTING BLOCK OF TSK AREA.
2765 02453 3336 DCA FTCHMS+2
2766 02454 4025 VCDF /FORCE CORRECT DF AND FLD IN CORE.
2767 02455 0000 MTFFLD, 0
2768 02456 4054 CALL /SEND MS TO SWAP TASK IN.
2769 02457 0107 SNDWTR+DFPARM
2770 02460 2534 FTCHMS /FORCE DATAFLD AS PARAMETER IN MS.
2771 02461 0004 SMTSDSK
2772 02462 7240 CLA CMA

```

```

2773 02463 3255 DCA MTFFLD /FLAG FIRST TIME THROUGH RELOCATION LOOP.
2774
2775 /THE TSK IS NOW SWAPPED IN.
2776 /DF STILL IS ITS FLD.
2777 /UPDATE RELOCATABLE CODE.
2778 02464 1334 TAD FTCHMS /GET COMPLETION STATUS
2779 02465 7440 SZA /MUST BE 0.
2780 02466 4576 SYSHLT
2781 02467 1335 TAD MTFPAG
2782 02470 3151 DCA MON1 /PTR TO TREATED PAGE.
2783 02471 1551 RLOCLP, TAD MON1 /ADD VALUE OF MTFPAG (FIRST PAGE) TO
2784 /INITIAL LOCS OF EACH PAGE.
2785 02472 7450 SNA /TREAT MORE LOCS ON THIS PAGE?
2786 02473 5301 JMP RLOC1 /NO.
2787 02474 1335 TAD MTFPAG
2788 02475 1146 TAD M200 /REMEMBER TSK OFFSET OF 200.
2789 02476 3551 DCA MON1
2790 02477 2151 ISZ MON1
2791 02500 5271 JMP RLOCLP
2792 02501 2255 RLOC1, SZ MTFFLD /FIRST TIME HERE?
2793 02502 5307 JMP RLOC2 /NO.
2794 02503 1151 TAD MON1
2795 02504 7001 IAC
2796 02505 0132 AND C177
2797 02506 3334 DCA FTCHMS /PERHAPS FUTURE PC.
2798 02507 1151 RLOC2, TAD MON1
2799 02510 0146 AND C7600
2800 02511 1133 TAD C200
2801 02512 3151 DCA MON1 /PTR TO NEXT PAGE
2802 02513 2331 ISZ MTFCT /MORE PAGES TO BE TREATED?
2803 02514 5271 JMP RLOCLP /YES.
2804
2805 /ALL RELOCATION DONE. UPDATE PC.
2806 02515 4060 GET;4 /FETCH OLD PC, IF ANY. (NOTE: DF:=0).
2807 02516 0004
2808 02517 7450 SNA /IS THERE AN OLD PC?
2809 02520 1334 TAD FTCHMS /IF NOT TAKE THE FIRST CODE LOCATION.
2810 02521 1335 TAD MTFPAG
2811 02522 3510 DCA X
2812 02523 7240 CLA CMA
2813 02524 1107 TAD BASE
2814 02525 6002 ICF /\
2815 02526 3155 DCA INT1 /\PTR TO TCB[4] IN INT1 FOR INRUNQ.
2816 02527 4774 JMS INRUNQ /\
2817 / ION /\INTERRUPTS WILL BE ON SOON.
2818 02530 5200 JMP MTFTIN
2819
2820 02531 0000 MTFCT, 0
2821
2822 02532 5051 MTFTCH
2823 02533 0000 0
2824 02534 0000 FTCHMS, 0 /FETCH MESSAGE.
2825 02535 0000 MTFPAG, 0
2826 02536 0000 0
2827

```

2828	02537	1703	AVPT;(0;AVPT=-2
2829	02540	2573	
2830		2537	
2831			
2832	02573	0000	
2833	02574	1076	
2834	02575	6000	
2835	02576	1221	
2836	02577	0537	
2837		2600	PAGE

```

2838      /MTIME.
2839      /SYSTEM TIMER.
2840      /KEEP TRACK OF SYSTEM TIME AND TIME OUT QUEUE.
2841      /THE TIME OUT Q (TOQ) IS BUILT OF 2-WORD TIMEOUT NODES.
2842      /LAYOUT OF TIME OUT NODE:
2843      /W0:   LINK TO NEXT NODE
2844      /W1:   PTR TO TCB[8] (AC) OF TASK;
2845      /      IF W1=0 THE TASK HAS BEEN REINSERTED IN RUNQ, AND THIS
2846      /      NODE MUST BE DELETED FROM TOQ.
2847      /
2848      /INSERTING A TSK IN TOQ:
2849      /1.   DELETE TSK FROM RUNQ.
2850      /2.   REQUEST 2-WORD NODE.
2851      /3.   LET W1 OF NODE PT TO TSKS AC, LINK NODE IN TOQ.
2852      /4.   LET TCB[5] PT TO NODE W1.
2853      /      THE LATTER IS NEEDED TO DELETE THE TSK FROM TOQ WHEN IT
2854      /      IS REINSERTED IN RUNQ (FOR WHATEVER REASON).
2855      /THE TIME OUT VALUE IS KEPT IN AC OF THE TASK.
2856      /EACH 0.1 SEC THE TIME OUT VALUE IS INCREMENTED. ON OVERFLOW THE
2857      /TASK IS REINSERTED IN THE RUNQ.
2858      /NOTE: EXCEEDING TIME OUT LIMITS IS THUS INDICATED BY AC=0 AT
2859      /RETURN.
2860
2861      /TIMELP,      CALL
2862      /              WTRP      /WAIT FOR CLOCK INTERRUPT.
2863      /              TIMEMS
2864      02600  7200  TIMEIN, CLA
2865      02601  2267          ISZ      TIME
2866      02602  5205          JMP      .+3
2867      02603  2270          ISZ      TIME+1
2868      02604  7000          NOP
2869      02605  1377          TAD      (TQHEAD /INSPECT TIME OUT Q.
2870      02606  3150          DCA      MON0 /PTR TO PRECESSOR.
2871      02607  1550  TOQLP, TAD      MON0 /\GET PTR TO NEXT NODE.
2872      02610  7450          SNA      /END REACHED?
2873      02611  5242          JMP      TOQEX
2874      02612  3010          DCA      X0 /PTR TO NODE.
2875      02613  6202          CIF      /\DEAF. POSSIBLY CHANGING TSK'S AC.
2876      02614  1410          TAD      X0 /\GET PTR TO TSKS AC.
2877      02615  7450          SNA      /\0 INDICATES TASK NO LONGER IN TOQ.
2878      02616  5230          JMP      TOQFN /FREE THIS NODE.
2879      02617  3151          DCA      MON1 /\PTR TO TSKS AC.
2880      02620  2551          ISZ      MON1 /\INCREMENT TIME OUT VALUE.
2881      02621  5240          JMP      TOQLP2 /NO OVERFLOW.
2882      02622  6002          ICF      /\OVERFLOW. INSERT TSK IN RUNQ.
2883      02623  1140          TAD      M4 /\
2884      02624  1151          TAD      MON1 /\
2885      02625  3155          DCA      INT1 /\PTR TO TCB[4] WAIT WORD.
2886      02626  4776          JMS      INRUNQ /\
2887      02627  6001          ION      /\
2888      02630  1550  TOQFN, TAD      MON0
2889      02631  3151          DCA      MON1 /PTR TO NODE TO BE DELETED
2890      02632  1551          TAD      MON1 /DELETE FROM TOQ
2891      02633  3550          DCA      MON0
2892      02634  1151          TAD      MON1 /PTR TO NODE IN AC

```

```

2893 02635 4572 JMS ZFN
2894 02636 4334 AVL2
2895 02637 5207 JMP TOQLP
2896 02640 1550 TOQLP2, TAD MON0
2897 02641 5206 JMP TOQLP-1
2898 02642 1375 TOQEX, TAD (-TIMEMS+2
2899 02643 3774 DCA MTTIME-3
2900 02644 7330 WTRPMT, AC4000
2901 02645 4773 JMS CURWT
2902 02646 5772 JMP SCED
2903
2904 /INSERT CURRENT TSK IN TOQ.
2905 02647 0000 TQIN, 0 /\
2906 02650 4771 JMS GN2 /\GET 2-WORD NODE.
2907 02651 1263 TAD TQHEAD /\
2908 02652 3410 DCA X0 /\
2909 02653 1010 TAD X0 /\
2910 02654 3263 DCA TQHEAD /\APPEND IN TOQ,
2911 02655 7325 AC3 /\
2912 02656 1164 TAD CURTSK /\
2913 02657 3410 DCA X0 /\PTR TO TSKS AC IN NODE W1.
2914 02660 1010 TAD X0 /\
2915 02661 3564 DCA CURTSK /\TCB[5] PT TO NODE W1.
2916 02662 5647 JMP TQIN /\
2917
2918 02663 0000 TQHEAD, 0 /HEAD OF TIME OUT QUEUE.
2919
2920 02664 5067 MTTIME /SENDER WORD,
2921 02665 0000 0 /LINK WORD
2922 02666 0000 TIMEMS, 0 /SOME STATUS MIGHT GO HERE.
2923 02667 0000 TIME, 0
2924 02670 0000 0
2925 IFDEF MULTIP <
2926 IFZERO SYTIME-MULTIP <
2927 /CONNECTED ROUTINE. COUNTS 55 MULTIPLEXER INTERRUPTS
2928 /BEFORE IT REPORTS TIMEMS.
2929 MULINTR,0 /!
2930 T1ON /!
2931 ISZ MULTCT /!
2932 JMP FSTEXT /!
2933 TAD M55 /!
2934 DCA MULTCT /!
2935 EXIT /!
2936 TIMEMS /!
2937
2938 MULTCT, -55
2939 M55, -55
2940 OCTAL
2941 >>

```

```

2942      /APP. A4. MC8.4. MONITOR CALLS AND RELATED STUFF.
2943      /***** MONITOR CALLS AND RELATED STUFF *****/
2944      /
2945      /BODY OF VIRTUAL READ DATAFIELD ROUTINE,
2946      /CALLED USING COMMON PART OF PAGE 0 AS FOLLOWS:
2947      /      JMS      VVRDF
2948      /
2949      /VVRDF, 0
2950      /      IOF
2951      /      CIF 0
2952      /      JMP      (VRDF1
2953      /NOTE: PRESERVE LINK, AC:=VFLD#, DF:=IF.
2954      02671 7630 VRDF1, SZL CLA      /!PRESERVE L.
2955      02672 7307          AC4          /!
2956      02673 6214          RDF          /!
2957      02674 7112          CLL RTR      /!
2958      02675 7010          RAR          /!ACT DF IN AC,
2959      02676 1170          TAD          ZCORMAP /!
2960      02677 3150          DCA          MON0 /!
2961      02700 6201          CDF 0        /!
2962      02701 1550          TAD      MON0 /!GET VFLD#
2963      02702 0130          AND          C77 /!REMOVE RUBBISH
2964      02703 3004          DCA          MONAC /!
2965      02704 1370          TAD      (VVRDF /!
2966      /FALL INTO RETURN JUMP
2967
2968      RTN,
2969      02705 3150          DCA          MON0 /!STORE PTR TO PC OF CALLING TSK.
2970      02706 1052          TAD          CDFUF+1 /!GET CDF TO CALLING FLD.
2971      02707 1120          TAD          C2 /!CHANGE INTO CIF CDF
2972      02710 3311          DCA          .+1 /!
2973      02711 6203          CIF CDF      /!TO CALLING FLD
2974      02712 1550          TAD      MON0 /!GET PROGRAM COUNTER
2975      02713 3150          DCA          MON0 /!
2976      02714 1004          TAD          MONAC /!
2977      02715 6001          ION          /!
2978      02716 5550          JMP      MON0
2979
2980
2981      /BODY OF MSNODE ROUTINE.
2982      /CALLED USING COMMON PART OF PAGE 0 AS FOLLOWS:
2983      /TO REQUEST A MESSAGE:
2984      /      CLA
2985      /      JMS VMSNODE
2986      /TO RETURN A MESSAGE:
2987      /      TAD (MS[2]
2988      /      JMS VMSNODE      /RETURN CONTENTS OF MS[2]
2989      /      /IN AC.
2990      /VMSNODE,0
2991      /      IOF
2992      /      CIF CDF 0
2993      /      JMP      XMSNOD1
2994      02717 7450 MSNOD1, SNA          /!REQUEST OR RETURN?
2995      02720 5330          JMP          MSNREQ /!REQUEST.
2996      02721 3150          DCA          MON0 /!RETURN. SET PTR TO MS[2]

```

```

2997 02722 7344      ACM2      /!
2998 02723 1150      TAD      MON0    /!GET PTR TO MS[0].
2999 02724 4572      JMS      ZFN      /!
3000 02725 4350      AVL5
3001 02726 1550      TAD      MON0    /!FETCH CONTENTS OF MS[2].
3002 02727 5335      JMP      MSNCD2  /!
3003 02730 4767' MSNREQ, JMS      GN5    /!REQUEST, X0 PTS TO MS[0]=-1
3004 02731 3410      DCA      X0      /!
3005 02732 3410      DCA      X0      /CLEAR MS[1] LINKWORD.
3006 02733 1010      TAD      X0      /!
3007 02734 7001      IAC          /!
3008 02735 3004 MSNOD2, DCA      MONAC  /!
3009 02736 1366      TAD      (VMSNODE/!GET PTR TO PC.
3010 02737 5305      JMP      RTN     /!RETURN JUMP INTO CALLING TSK.
3011
3012 /HELP ROUTINE FOR NORMAL MONITOR CALLS.
3013 /GET SECOND ARGUMENT.
3014 02740 0000      INARG2, 0
3015 02741 4051      JMS      CDFUF
3016 02742 1454      TAD      MONPC
3017 02743 6201      CDF 0
3018 02744 2054      ISZ      MONPC
3019 02745 3162      DCA      MARG2
3020 02746 5740      JMP      INARG2
3021
3022 02747 2537      AVPT;(0;AVPT=-2
3023 02750 2765
3024 2747
3025
3026 02765 0000
3027 02766 0014
3028 02767 4346
3029 02770 0045
3030 02771 4332
3031 02772 0617
3032 02773 0545
3033 02774 5064
3034 02775 5114
3035 02776 1076
3036 02777 2663
3037 3000 PAGE
3038
3039

```

```

3040      /MONITOR ENTRY.
3041      /CALLED USING COMMON PART OF PAGE 0 AS FOLLOWS:
3042      /          JMS      VCALL
3043      /          ....
3044      /VCALL, 0
3045      /          CIF 0
3046      /          IOF
3047      /          JMP      (MONITOR
3048      /NOTE: A POSSIBLE SCDREQ MAY HAVE SET SCDREQ TO 0.
3049
3050      /SCHEDULING IS INHIBITED DURING MONITOR CALLS BY SETTING SCDINH
3051      />0. A SCDREQ IS SIGNALLED BY CLEARING SCDREQ. AT
3052      /END OF EACH MONITOR CALL SCDREQ IS TESTED TO SEE WETHER
3053      /SCHEDULING IS REQUIRED (SEE MONEX AND MONOUT).
3054      /TWO TYPES OF MONITOR CALLS ARE DISTINGUISHED:
3055      /NORMAL CALLS: CALLS DIRECTLY FROM A TASK
3056      /SPECIAL CALLS: SOME PAGE 0 SERVICE ROUTINE CALLED THE MONITOR.
3057      /PC IS USED TO DISTINGUISH BETWEEN THESE CALLS:
3058      /PC<VERHLT+1 CALL FROM VRCDF
3059      /PC=VERHLT+1 CALL FROM VERHLT
3060      /PC>VERHLT+1
3061      / <100 CALL FROM VRCDF
3062      /PC>100 CALL FROM TASK
3063
3064      03000 3004 MONITOR,DCA MONAC /-
3065      IFNDEF PDP8E <
3066      RDF /-
3067      DCA MON0 /-SAFE DF FOR CDFINS,
3068      >
3069      IFDEF PDP8E <
3070      03001 6004 GTF /-GET L +GTFL.
3071      03002 0001 AND X6000 /-
3072      03003 7004 RAL
3073      >
3074      03004 7006 RTL /-TRICK: LINK TO B10
3075      03005 6214 RDF /-
3076      03006 7112 CLL RTR;RAR /-
3077      03007 7010
3078      03010 1167 TAD SCDINH /-ADD (OVERRIDE PRIO SCED)=CONDITION.
3079      03011 2167 ISZ SCDINH /-INHIBIT SCHEDULING DURING MONCALLS.
3080      03012 6001 ICN /-SCHEDULING IS NOW INHIBITED.
3081      03013 4051 JMS CDFUF /CDF CALLING FLD
3082      03014 6214 RDF /GET IN INSTRUCTION FIELD.
3083      03015 3007 DCA MONFL
3084      03016 1777 TAD I (VCALL
3085      03017 3054 DCA MONPC /STORE PC OF CALLING TASK.
3086      03020 1054 TAD MONPC
3087      03021 0145 AND C7700
3088      03022 7640 SZA CLA
3089      03023 5255 JMP MONIT2 /NORMAL CALL.
3090      03024 1054 TAD MONPC /SPECIAL CALL, WHICH ONE?
3091      03025 1376 TAD (-VERHLT-2
3092      03026 7450 SNA
3093      03027 4576 SYSHLT /ERHLT
3094      03030 7710 SPA CLA

```

```

3095 03031 5242      JMP      VRCDF1 /VRCDF
3096
3097                VCDF1,      /VCDF.
3098 03032 1775      TAD I      (VVRCDF /WE SAVED AC THERE
3099 03033 3004      DCA      MONAC
3100 03034 1774      TAD I      (VVCDF /GET PC OF CALLING TASK,
3101 03035 3054      DCA      MONPC
3102 03036 1773      TAD I      (VCDFTM /GET PTR IN FLDTABLE
3103 03037 6201      CDF 0
3104 03040 3153      DCA      MON3
3105 03041 5772'     JMP      FCSWAP /ISSUE A SWAPREQUEST IF POSSIBLE,
3106
3107                /DO CDF TO REQUESTED FIELD.
3108                /CALL OF MINIMUMTIME CDF ROUTINE.
3109                /IF THE REQUESTED FLD IS IN CORE, STORE ITS CDF IN CALLING FLD.
3110 03042 7040      VRCDF1, CMA
3111 03043 1775      TAD I      (VVRCDF /GET PC
3112 03044 3054      DCA      MONPC
3113
3114                /WE SAVED THE PC BEFORE THE CALL OF
3115                /VRCDF. SO THE FIRST INSTR EXECUTED BY
3116                /THE TSK WILL BE: CALLING VRCDF,
3117                /IF THE REQUESTED FLD IS NOT IN CORE, TSK
3118                /IS SAVED WITH REQUESTED DF, AND THIS
3119                /EXTRA CALL WILL DO A CHECK WHICH WOULD
3120                /PROBABLY OCCUR SOON ANYHOW.
3121                /IF REQUESTED DF IN CORE, WE USE THIS
3122                /EXTRA CALL TO SET THE PROPER DF, WHICH
3123                /COSTS NO TIME.
3124 03045 6201      CDF 0
3125 03046 1164      TAD      CURTSK /GET REQUESTED DATAFIELD
3126 03047 1124      TAD      C10
3127 03050 3010      DCA      X0      /PTR TO TCB[14]; REQCDF.
3128 03051 1410      TAD I      X0
3129 03052 4771'     JMS      FCHECK /CHECK IF IT IS IN CORE
3130                /IF NOT THE TSK IS SAVED WITH THE NEW DATAFIELD.
3131 03053 4051      JMS      CDFUF /IT IS IN CORE.
3132 03054 5770'     JMP      VRCDF3 /STORE THE CDF IN CALLING FLD
3133                /AND FALL INTO MONEX.

```

```

3132      /NORMAL MONITOR CALLS.
3133      /CALLING SEQUENCE IN TASK:
3134      /      CALL
3135      /      MONFUNC /FUNCTIONWORD
3136      /      ...      /POSSIBLE ARGS.
3137      /      ...      /POSSIBLE ERROR RETURN.
3138      /      ...      /NORMAL RETURN.
3139      /
3140      /THERE ARE ABOUT 25 DIFFERENT MONCALLS DISTINGUISHED BY THE
3141      /FUNCTIONWORD. THE CALL IS SPECIFIED IN B6-10 OF THE FUNCTION
3142      /WORD. B11 OF THE FUNCTIONWORD IS SET IF THE CALL USES
3143      /ARGUMENTS.
3144
3145      /SOME CALLS ACCEPT OPTIONS THAT ARE SPECIFIED IN THE OTHER BITS
3146      /OF THE FUNCTIONWORD.
3147      /
3148      /LIST OF MONITORCALLS.
3149      0200 STALL=0+2+TIMEOUT
3150      /      OPTIONS: SWPOUT.
3151      /      INSERT TSK IN TIMEOUT QUEUE (SEE TIMEOUT OPTION).
3152      0003 SNDMS=1+2+1 /((SEND A MESSAGE)
3153      /      OPTIONS: NONREP, DFPARM.
3154      /      SEND MS TO A TSK.
3155      /      ARG1: PTR TO MS[2]
3156      /      ARG2: STSK# OF ADDRESSED TSK.
3157      0005 WTRP=2+2+1 /((WAIT FOR REPORT).
3158      /      OPTIONS: TIMEOUT, SWPOUT.
3159      /      EITHER SPECIFIC WAIT, (I.E., WAIT FOR THE REPORT ON THE
3160      /      MS PTED TO BY ARG1),
3161      /      OR GENERAL WAIT REPORT (WAIT FOR THE FIRST INCOMING
3162      /      REPORT).
3163      /      AT RTN AC =PTR TO MS[2] OF REPORTED MS, L=1.
3164      /      ARG1: SPECIFIC WAIT; PTR TO MS[2], GENERAL WAIT: 0.
3165      0007 SNOWTR=3+2+1
3166      /      OPTIONS: AS IN 1 AND 2.
3167      /      SEND MESSAGE AND WAIT FOR ITS REPORT.
3168      /      ARG1: PTR TO MS[2] OF SENT MS.
3169      /      ARG2: STSK# OF ADDRESSED TSK.
3170      0013 RP=5+2+1 /((REPORT A MESSAGE).
3171      /      REPORT A MS TO ITS SENDER (I.E., INSERT IT IN ITS RPQ),
3172      /      IF MS[0]B0=0 NO ONE IS WAITING FOR THE REPORT,
3173      /      SO THE MS CAN IMMEDIATELY BE FREED.
3174      /      ARG1: PTR TO MS[2] OF REPORTED MS.
3175      0014 WTMS=6+2 /((WAIT FOR MESSAGE).
3176      /      OPTIONS: KEEP, TIMEOUT, SWPOUT.
3177      /      WAIT UNTIL A MS IS SENT TO THIS TSK.
3178      /      IF KEEP OPTION SPECIFIED, KEEP TSK CLAIMED:
3179      /      I.E., WAIT FOR MS SENT BY THE SAME TASK AS BEFORE
3180      /      AT RTN: AC=PTR TO MS[2] OF RECEIVED MS, L=0.
3181      0016 DISINTR=7+2 /((DISCONNECT FROM INTERRUPT).
3182      /      DISABLE DEVICE INTERRUPT.
3183      /      CRITICAL SYNCHRONIZATION!!! PATCH THE SKPCHN SUCH
3184      /      THAT FLAGS OF THE SPECIFIED DEVICE ARE NOT
3185      /      TREATED AS INTERRUPTS.
3186      /      CLEARS AC.

```

/APP, A4. MC8,4. MONITOR CALLS AND RE PAL8-V9H 02/14/77 PAGE 40-1

3187

/

AC=SKIPLOT.

```

3188      0021  CNINTR=10↑2+1  /((CONNECT A ROUTINE TO INTERRUPTSLOT)
3189      /      A ROUTINE TO TREAT INTERRUPTS OF THE SPECIFIED DEVICE IS
3190      /      ATTACHED TO THE DEVICES INTSLOT. THE ROUTINE MUST
3191      /      BE CORERESIDENT AND RESIDE IN THE CALLER'S DF.
3192      /      DENOTE CURTSK AS SENDER IN THE COMMUNICATION MS, IF ANY.
3193      /      CLEARS AC.
3194      /      AC=SKIPIOT.
3195      /      ARG1:  PTR TO ENTRY POINT OF ROUTINE TO BE CONNECTED.
3196      /      ARG2:  PTR TO MS[2] OF COMMUNICATION MS, IF ANY,
3197      /      0 OTHERWISE.
3198      0023  CLINTR=11↑2+1  /((CLAIM AN INTSLOT)
3199      /      ATTACH A MS TO THE SPECIFIED INTSLOT. THE MS IS REPORTED
3200      /      TO THE TSK, EACH TIME AN INTERRUPT ON THAT DEVICE
3201      /      OCCURS.
3202      /      CLEARS AC.
3203      /      AC=SKIPIOT.
3204      /      ARG1:  CLEARIOT
3205      /      ARG2:  PTR TO MS[2] OF MS TO BE ATTACHED.
3206      0025  FRINTR=12↑2+1  /((FREE INTSLOT)
3207      /      SET INTSLOT OF SPECIFIED DEVICE IN DISCARD MODE (I.E.,
3208      /      INTERRUPTS OF THAT DEVICE ARE DISCARDED).
3209      /      CLEARS AC.
3210      /      AC=SKIPIOT.
3211      /      ARG1:  CLEARIOT (TO CLEAR THE DEVICEFLAG).
3212      0027  SIMINTR=13↑2+1 /((SIMULATE INTERRUPT)
3213      /      ENTER A DEAF SECTION THAT MUST BE LEFT BY
3214      /      "CIF 0; IEXIT; 0". THIS IS MOST OFTEN USED TO
3215      /      START UP LOOPING CONNECTED ROUTINES. WHEN THE
3216      /      RETURN FROM INTERRUPT "IEXIT" IS TAKEN,
3217      /      THE CALLER IS CONTINUED.
3218      /      THE DEAF SECTION MUST RESIDE IN CALLER'S DF.
3219      /      LEAVES AC UNCHANGED.
3220      /      ARG1:  ENTRYPOINT OF DEAF SECTION.
3221      0031  REQ PAG=14↑2+1  /((REQUEST PAGES).
3222      /      1-37 CONSECUTIVE PAGES OF CORE ARE REQUESTED.
3223      /      AT RTN: AC B0-4 =PAGE# OF FIRST PAGE,
3224      /      B6-11 =VFLD#.
3225      /      ARG1:  B7-11 LENGTH (1<=LENGTH<=37), B5 CORERESIDENT.
3226      0033  RTN PAG=15↑2+1 /((RETURN PAGES PREVIOUSLY REQUESTED USING
3227      /      REQ PAG).
3228      /      CLEARS AC.
3229      /      AC B0-4 =PAGE# OF FIRST PAGE, B6-11 =VFLD#.
3230      /      ARG1:  B7-11 LENGTH (1<=LENGTH<=37), B5 CORERESIDENT.
3231      0034  REQ FLD=16↑2  /((REQ WHOLE FLD FOR DATA STORAGE).
3232      /      THE VFLD# IS RETURNED IN AC B6-11.
3233      0036  RTN FLD=17↑2  /((RETURN FLD PREVIOUSLY REQUESTED USING REQ FLD).
3234      /      CLEARS AC.
3235      /      AC B6-11 =VFLD#.
3236      0041  REQ STL=20↑2+1 /((REQUEST ENTRY IN STATIC TASK LIST).
3237      /      A FREE ENTRY IN THE STATIC TSK LIST IS SEARCHED, ARG1 AND
3238      /      ARG2 ARE ASSIGNED TO IT.
3239      /      NOTE: THE ENTRY IS FREED AGAIN BY CLEARING
3240      /      ITS FIRST WORD (NO SPECIAL MONITORCALL!!).
3241      /      AT RETURN AC=STSK#.
3242      /      ARG1:  CONTENTS OF TCB[15] (>0).

```

```

3243          /      ARG2:  B7-11 LENGTH, B5 TSK CORERESIDENT, B3 TSK USES
3244          /      OTO PART OF PAGE 0, B0=2 TSKS PRIO.
3245 0042 REQTCB=21+2      /(ATTACH TCB TO ENTRY IN STL).
3246          /      SET UP TCB USING THE SPECIFIED ENTRY IN STL.
3247          /      AC=STSK#.
3248          /      AT RETURN AC=PTR TO TCB[5].
3249 0044 RTNTCB=22+2      /(RETURN TCB).
3250          /      DETACH TCB FROM ENTRY IN STL.
3251          /      NOTE!!!! TSK MUST BE STOPPED BEFORE!!!!.
3252          /      CLEARS AC.
3253          /      AC=STSK#.
3254 0046 LOCK=23+2      /(LOCK FLD IN CORE)
3255          /      LOCK THE CURRENT DATAFIELD IN CORE.
3256          /      LEAVES AC UNCHANGED.
3257 0050 UNLOCK=24+2      /(UNLOCK FLD)
3258          /      UNLOCK THE CURRENT DATAFIELD.
3259          /      LEAVES AC UNCHANGED.
3260 0052 REQCDF=25+2      /(REQUEST DATAFLD).
3261          /      SET UP FOR USING THE MINIMUMTIME CDF ROUTINE VRCDF.
3262          /      SUBSEQUENT CALLS ON VRCDF WILL CHANGE DF TO THE
3263          /      REQUESTED DF.
3264          /      LEAVES AC UNCHANGED.
3265          /      AC B6-11:  REQUESTED DF.
3266 0054 EXIT=26+2      /(DISMISS TSK AND TCB FROM CORE).
3267          /      THE CORE OCCUPIED BY THE TSK IS RETURNED, AS IS ITS TCB.
3268          /      THE TSK IS RESTARTED FROM SCRATCH BY SENDING A MS
3269          /      TO IT.
3270 0056 STOP=27+2      /(STOP A TSK).
3271          /      A TSK IS STOPPED. IT WON'T BE RERUN UNTIL THE RESUME
3272          /      COMMAND IS GIVEN FOR IT. IT IS REMOVED FROM THE
3273          /      TIMEOUT QUEUE. ALL OTHER WAITFUNCTIONS ARE
3274          /      NORMALLY PERFORMED AS LONG AS IT IS STOPPED.
3275          /      CLEARS AC.
3276          /      AC=STSK#.
3277 0060 RESUME=30+2      /(RESUME A TSK).
3278          /      RESUME A TSK PREVIOUSLY STOPPED BY THE STOP COMMAND.
3279          /      THE TSK WILL BE RESUMED AS SOON AS OTHER
3280          /      WAITFUNCTIONS EXPIRE, IF NO WAITS WERE IN, THE
3281          /      TSK IS RESUMED IMMEDIATELY.
3282          /      CLEARS AC.
3283          /      AC=STSK#.
3284 0065 FILTCB=32+2+1      /(FILL A TCB)
3285          /      AC=STSK#
3286          /      CLEARS AC.
3287          /      ARG1=PTR TO CONTENTS OF TCB.

```

```

3288      /LIST OF OPTIONS.
3289      4000  CHKONLY=4000
3290      4014  CHRCQ=WTMS+CHKONLY /CHECK RECEIVE Q.
3291      /      EQUAL TO WTMS. BUT IF THE TASK SHOULD BE SET TO WAIT IT
3292      /      IS INSTEAD CONTINUED WITH AC=0.
3293      4005  CHRPQ=WTRP+CHKONLY /CHECK REPORT Q.
3294      /      EQUAL TO WTRP. BUT IF THE TASK SHOULD BE SET TO WAIT, IT
3295      /      IS CONTINUED WITH AC=0.
3296      2000  NONREP=2000      /THE MS SENT NEED NOT BE REPORTED
3297      /      ((CLEAR MS[0] B0).
3298      1000  KEEP=1000      /WAIT FOR MS WITHOUT CLEARING THE CLAIM WORD,
3299      /      (( TCB[0]) SO THAT WE WAIT FOR A MS OF THE
3300      /      SAME TASK.
3301      0400  SWPOUT=400      /SWAP TSK OUT. THE CORE OCCUPIED BY
3302      /      THE TSK IS FREED AGAIN, BUT THE TCB IS KEPT, AS
3303      /      SOON AS THE TSK BECOMES RUNNABLE A FRESH COPY IS
3304      /      FETCHED FROM DISK.
3305      0200  TIMEOUT=FWTTM
3306      0200  TIMEOUT=200      /INSERT TSK IN TIMEOUT Q. AS LONG AS THE TSK IS
3307      /      NOT RUNNABLE EACH 0.1 SEC AC WILL BE
3308      /      INCREMENTED.
3309      /      ON OVERFLOW THE TSK IS FORCED IN THE RUNQ (WITH
3310      /      AC=0!)
3311      0100  DFARM=100      /STORE ACT FLD# OF CURRENT DF IN
3312      /      MS[2] B6-8.

```

```

3313      /NORMAL CALL.
3314      /DF IS CALLING FLD.
3315      03055 1454 MONIT2, TAD 1 MONPC /GET FUNCTION WORD
3316      03056 3160 DCA MONFUNC
3317      03057 2054 ISZ MONPC
3318      03060 1454 TAD 1 MONPC
3319      03061 3161 DCA MARG1 /STORE FIRST ARG.
3320      03062 6201 CDF 0
3321      IFNDEF PDP8E < /UPDATE CDFINS
3322      TAD MON0
3323      TAD CCDF
3324      DCA CDFINS
3325      >
3326      IFDEF STATX <
3327      03063 1160 TAD MONFUNC
3328      03064 0272 AND MON76 /2*FUNCTIONCODE
3329      03065 1367 TAD (MONSTAT-1
3330      03066 3010 DCA X0 /PTR IN MONITOR STATISTICS.
3331      03067 2410 ISZ 1 X0
3332      03070 5273 JMP .+3
3333      03071 2410 ISZ 1 X0
3334      03072 0076 MON76, 76 /INNOCENT AND.
3335      >
3336      03073 1160 TAD MONFUNC
3337      03074 7010 RAR
3338      03075 0125 AND C37
3339      03076 1303 TAD MONJMP
3340      03077 3302 DCA .+3
3341      03100 7430 SZL /ARGUMENTS USED?
3342      03101 2054 ISZ MONPC /YES.
3343      03102 5000 JMP
3344
3345      03103 5704 MONJMP, JMP 1 .+1
3346      03104 3315 XSTALL /INSERT IN TIMEOUT QUEUE.
3347      03105 3200 XSNOMS /SEND MESSAGE
3348      03106 3236 XWTRP /WAIT FOR REPORT
3349      03107 3200 XSNOWTR /SEND MESSAGE AND WAIT FOR ITS REPORT.
3350      03110 3144 XERROR
3351      03111 3251 XRP /REPORT A MESSAGE
3352      03112 3303 XWTMS /WAIT UNTIL A MESSAGE IS RECEIVED
3353      03113 3677 XDISINTR /DISCONNECT A DEVICE FROM INTERRUPT
3354      03114 3701 XCNINTR /CONNECT A ROUTINE TO INTSLOT
3355      03115 3714 XCLINTR /CLAIM (ATTACH A MS TO) AN INTSLOT
3356      03116 3737 XFRINTR /FREE INTSLOT (BACK INTO DISCARD MODE)
3357      03117 3741 XSIMINTR /SIMULATE INTR (START UP LOOPING
3358      /CONNECTED ROUTINE)
3359      03120 4042 XREQPAG /REQUEST A NUMBER OF CONSECUTIVE PAGES
3360      03121 4043 XRTNPAG /RETURN A NUMBER OF CONSECUTIVE PAGES
3361      03122 4051 XREQFLD /REQUEST FLD FOR DATA STORAGE
3362      03123 4060 XRTNFLD /RETURN FLD
3363      03124 4132 XREQSTL /REQUEST ENTRY IN STL AND SET ITS
3364      /CONTENTS.
3365      03125 4160 XREQTCB /ATTACH TCB TO ENTRY IN STL.
3366      03126 4034 XRTNTCB /DETACH TCB FROM ENTRY IN STL AND FREE
3367      /IT.

```

3368	03127	4201	XLOCK	/LOCK DF IN CORE
3369	03130	4200	XUNLOCK	/UNLOCK DF
3370	03131	3600	XREGCDF	/SET UP FOR USING VRCDF ROUTINE
3371	03132	3267	XEXIT	/DISMISS TSK AND TCB FROM CORE
3372	03133	3630	XSTOP	/STOP A TSK.
3373	03134	3647	XRESUME	/RESUME A TSK.
3374	03135	3144	XERROR	/PROCES IDENTIFY.
3375	03136	3466	XFILTCB	/FILL TCB
3376	03137	3144	XERROR	
3377	03140	3144	XERROR	
3378	03141	3144	XERROR	
3379	03142	3144	XERROR	
3380	03143	3144	XERROR	
3381			IFNZRO .-MONJMP-41 <ALLOCATION ERROR!>	
3382				
3383				
3384	03144	4576	XERROR, SYSHLT	
3385				
3386	03145	2747	AVPT;(0;AVPT=.-2	
3387	03146	3166		
3388		3145		
3389				
3390	03166	0000		
3391	03167	4731		
3392	03170	0726		
3393	03171	2000		
3394	03172	2010		
3395	03173	0042		
3396	03174	0025		
3397	03175	0020		
3398	03176	7753		
3399	03177	0054		
3400		3200	PAGE	

```

3401      /***** MESSAGE EXCHANGE *****/
3402      /
3403      /TASKS ARE SYNCHRONIZED AND COMMUNICATE TO ONE ANOTHER BY SENDING
3404      / AND REPORTING MESSAGES.
3405      /A MESSAGE CONSISTS OF 5 CONSECUTIVE LOCATIONS IN FLD 0,
3406      /MESSAGE LAYOUT:
3407      /W0      SENDERWORD.
3408      /          B0=0: MESSAGE NEED NOT BE REPORTED.
3409      /          B0=1: MESSAGE MUST BE REPORTED.
3410      /          B1-11 BIT 1-11 OF TCBPTR OF SENDER (B0 OF TCBPTR ALWAYS
3411      /              SET!).
3412      /W1      LINKWORD (USED TO LINK THE MS IN RCQ OR RPQ).
3413      /W2-4     USER DEFINED INFORMATION,
3414      /MESSAGES ARE DRAWN FROM A POOL OF STORAGE AND !MUST! BE RETURNED
3415      /THERE AS SOON AS THEY ARE NO LONGER USED.
3416      /
3417      /REQUESTING AND RETURNING MESSAGES.
3418      /A TASK REQUESTS A MESSAGE BY EXECUTING THE "MSREQ"-INSTRUCTION
3419      /      WITH AC=0!.
3420      /      THE INSTRUCTION IS EXECUTED USING THE GENERAL PART OF
3421      /      PAGE 0. IT RETURNS A PTR TO MS[2] IN AC.
3422      /A TASK RETURNS A MESSAGE BY EXECUTING THE "MSFREE"-INSTRUCTION
3423      /      WITH AC=PTR TO MS[2].
3424      /      THE INSTRUCTION ALSO USES THE GENERAL PAGE 0.
3425      /      IT RETURNS THE MESSAGE TO THE POOL OF STORAGE IN FLD0
3426      /      AND RETURNS THE CONTENTS OF MS[2] IN AC. SO THE
3427      /      INSTRUCTION CAN SIMULTANEOUSLY BE USED TO FETCH INFO FROM
3428      /      THE MS.
3429      /IN PRINCIPLE A TASK SENDS A MESSAGE TO ANOTHER TASK THEREBY
3430      /ACTIVATING THE OTHER TASK. WHEN THE SECOND TASK IS DONE IT
3431      /REPORTS THE MESSAGE. THE SENDER MAY WAIT FOR THE REPORT, CHECK
3432      /THE CONTENTS OF THE MESSAGE, THUS KNOWING THAT THE MS WAS
3433      /ACCEPTED AND THE JOB PROPERLY EXECUTED.
3434      /BELOW *A AND *B INDICATE SEPARATE TASKS. *A EXECUTES THE
3435      /MONITORCALL. THE FOLLOWING MONITORCALLS HAVE TO DO WITH
3436      /MESSAGES:
3437      /SNDMS SEND MESSAGE.
3438      /      *A SENDS A MS TO *B.
3439      /      ARG1=PTR TO MS[2]. ARG2=STSK# OF *B.
3440      /      WHEN A TASK IS FULLY INACTIVE (TASK ON DISK, AND NO TCB
3441      /      IN CORE) A SNDMS COMMAND MAY ACTIVATE IT. THE STSK# OF *B
3442      /      IDENTIFIES THE ENTRY IN THE STL CORRESPONDING TO *B.
3443      /      THE MONITOR CHECKS WETHER A TCB OF *B IS IN CORE, IF IT
3444      /      IS NOT IT ALLOCATES ONE FOR *B.
3445      /      SUBSEQUENT IT DENOTES THE TCBPTR OF *A IN MS[0],
3446      /      BUT BIT0 IS CLEARED AGAIN IF THE NONREP OPTION WAS
3447      /      SPECIFIED. FINALLY *B "RECEIVES" THE MS.
3448      /WTRP WAIT FOR REPORT.
3449      /      WAIT FOR A MESSAGE TO BE REPORTED.
3450      /      IF ARG1=0 ANY REPORT THAT IS RECEIVED WILL REACTIVATE THE
3451      /      TASK, OTHERWISE ARG1 IS A PTR TO MS[2] OF THE MS WHOSE
3452      /      REPORT IS ALLOWED TO REACTIVATE THE TSK.
3453      /SNDWTR SEND MS AND WAIT FOR ITS REPORT.
3454      /      JUST A COMBINATION OF THE TWO FORMER CALLS.
3455      /RP      REPORT ON A MS.

```

```

3456      /      ARG1= PTR TO MS[2] OF THE REPORTED MS.
3457      /      THE MONITOR LOOKS IN THE SENDERWORD (MS[0]) TO FIND OUT
3458      /      WHICH TASK SENT THE MS. IF MS[0]B0 =0 NO TSK IS
3459      /      INTERESTED IN THE REPORT AND THE MS IS RETURNED TO THE
3460      /      POOL OF STORAGE IN FLD0, ELSE THE TSK THAT SENT THE MS
3461      /      "RECEIVES" ITS REPORT.
3462      /WTMS   WAIT FOR MS.
3463      /      THE TASK IS REACTIVATED AS SOON AS IT "RECEIVES" AN
3464      /      APPROPRIATE MS, NORMALLY THIS CALL DECLAIMS THE TSK, IT
3465      /      IS AGAIN READY TO RECEIVE MSS FROM ANY TASK, AS SOON
3466      /      AS IT RECEIVES ONE, IT IS CLAIMED BY THAT TASK,
3467      /      IF THE KEEP OPTION IS SPECIFIED THE TSK IS NOT DECLAIMED.
3468      /      IT THEN ONLY ACCEPTS MSS FROM THE TASK THAT CLAIMS IT,
3469      /
3470      /RECEIVING MESSAGES AND REPORTS.
3471      /IN THE TCB HEADS OF TWO QUEUES ARE ALLOCATED,
3472      /ONE IS THE RECEIVE Q (HEAD IN TCB[1]), THE OTHER IS THE
3473      /REPORT Q (HEAD IN TCB[3]), BESIDES THIS TCB[0]
3474      /IDENTIFIES THE CLAIMING TASK (SEE KEEP OPTION)
3475      /AND TCB[2] HOLDS A PTR TO THE REPORT THAT IS ALLOWED TO
3476      /REACTIVATE THE TSK (SEE WTRP).
3477      /IF WTMS OR WTRP IS EXECUTED, FIRST THE APPROPRIATE QUEUES ARE
3478      /INSPECTED. IF NO MESSAGE IN THESE QUEUES IS ALLOWED TO
3479      /REACTIVATE THE TASK, THEN THE WAITCONDITION IS SET, OTHERWISE
3480      /THE MSPTR OF THAT MS IS DENOTED IN THE TASK'S AC AND THE TASK
3481      /IS CONTINUED.
3482      /IF A MS IS SENT OR REPORTED, FIRST WE CHECK WETHER IT IS ALLOWED
3483      /TO REACTIVATE THE RECEIVER, IF IT IS THE RECEIVER'S
3484      /WAITCONDITION IS ERASED AND THE MSPTR DENOTED IN ITS AC.
3485      /OTHERWISE SENT MSS ARE ADDED TO THE RECEIVE Q
3486      /AND REPORTED MSS ARE ADDED TO THE REPORT Q.

```

```

3487          /SEND MESSAGE
3488          XSNDS,
3489          /SEND MS AND WAIT FOR REPORT.
3490          /ARG1=PTR TO MS[2]
3491          /ARG2=STSK# OF RECEIVER.
3492      03200  4777' XSNWTR,JMS      INARG2
3493      03201  1131          TAD      C100;DFPARM=100
3494          0100
3495      03202  0160          AND      MONFUNC /DFPARM OPTION SPECIFIED?
3496      03203  7640          SZA CLA
3497      03204  1007          TAD      MONFL /IF IT IS, DENOTE ACT DF IN MS.
3498      03205  0123          AND      C7 /GET DF
3499      03206  7106          CLL RTL;RAL
3500      03207  7004
3501      03210  1561          TAD I    MARG1
3502      03211  3561          DCA I    MARG1 /INSERT IN MS[2]
3503
3504          XSN2,
3505      03212  1162          TAD      MARG2 /STSK# OF RECEIVER
3506      03213  4776'          JMS      GETTCB /TCB[5] OF RECEIVER
3507      03214  1135          TAD      M1
3508      03215  6002          IOF
3509      03216  3155          DCA      INT1 /\TCB[4] OF RECEIVER.
3510      03217  7344          ACM2
3511      03220  1161          TAD      MARG1 /\
3512      03221  3154          DCA      INT0 /\PTR TO MS[0]
3513      03222  7332          AC2000;NONREP=2000/\
3514          2000
3515      03223  0160          AND      MONFUNC /\NONREP SPECIFIED?
3516      03224  7164          STL CMA RAL /\3777 IF NONREP, 7777 ELSE
3517      03225  0164          AND      CURTSK /\
3518      03226  3554          DCA I    INT0 /\
3519      03227  1155          TAD      INT1 /\
3520      03230  4571          JMS I    ZSNDRP /\SEND TO RECEIVER.
3521      03231  6001          ION
3522      03232  7307          AC4
3523      03233  0160          AND      MONFUNC /WAS IT SNWTR?
3524      03234  7650          SNA CLA
3525      03235  5775'          JMP      MONEX /NO.
3526
3527          /WAIT FOR REPORT.
3528          /IF ARG1=0 ANY REPORT IS ACCEPTED, OTHERWISE ARG1=PTR TO MS[2]
3529          /OF WAITED REPORT.
3530          XCHRPQ,
3531      03236  7346          XWTRP, ACM3 /LET MON0 PT TO TCB[2] WAITED REPORT WORD.
3532      03237  1164          TAD      CURTSK
3533      03240  3150          DCA      MON0
3534      03241  1161          TAD      MARG1
3535      03242  7041          CIA
3536      03243  7440          SZA
3537      03244  1120          TAD      C2
3538      03245  3550          DCA I    MON0 /INDICATE EITHER ZERO OR MINUS PTR TO
3539      03246  4774'          JMS      SMSG /MS[0]. SEARCH RPQ FOR THIS MS.
3540      03247  4000          4000
3541          /AC4000 /MS NOT IN RPQ. SET TASK TO WAIT.

```

```

3542 03250 5316      JMP      TSWAIT
3543
3544
3545
3546      /REPORT A MS.
3547      /ARG1=PTR TO MS[2].
3548 03251 7344      XRP,      ACM2
3549 03252 1161      TAD      MARG1
3550 03253 6002      IOF
3551 03254 3154      DCA      INT0      /\CALLING SNDREP.
3552 03255 1554      TAD      INT0      /\PTR MS[0]
3553 03256 7700      SMA      CLA      /\GET CONTENTS OF SENDERWORD MS[0].
3554 03257 5263      JMP      XRP2      /\ANYONE INTERESTED IN THIS REPORT?
3555 03260 4571      JMS      ZSNDREP /\NO.
3556 03261 6001      MONEX0, ION      /\YES. REPORT IT.
3557 03262 5775      JMP      MONEX      /\
3558 03263 1154      XRP2, TAD      INT0      /\FREE THE MS.
3559 03264 4572      JMS      ZFN      /\
3560 03265 4350      AVLS      /\
3561 03266 5261      JMP      MONEX0 /\
3562
3563      /EXIT.
3564      /FULL TASK DISMISS FROM CORE.
3565      /A COMPLETELY FRESH COPY IS SWAPPED IN AS SOON AS A MS IS SENT TO
3566      / THE TSK, BEFORE THAT TCB IS RETURNED.
3567 03267 3166      XEXIT, DCA      SCDREQ /FORCE SCHEDULING EVEN IF THERE IS A MS
3568      /FOR THIS TASK.
3569 03270 4773      JMS      RTNTSK /DISMISS TASKS CODE FROM CORE.
3570 03271 3054      DCA      MONPC /SET PC TO START OVER AGAIN.
3571 03272 4060      GET;      -4      /\ANY MS IN RCQ, (BASE PTED TO TCB[5]).
3572 03273 7774
3573 03274 7640      SZA      CLA
3574 03275 5303      JMP      XWTMS      /\YES
3575 03276 4772      JMS      CURWT      /\REMOVE CURTSK FROM RUNQ.
3576 03277 7040      CMA      /\
3577 03300 1162      TAD      MARG2 /\
3578 03301 4771      JMS      RNTCB /\AC PTS AT STL[TSK,0].
3579 03302 5770      JMP      SCED      /\
3580
3581      /WAIT FOR MS.
3582      /NO ARGUMENTS.
3583      /IF NO KEEP OPTION CLEAR TCB[0]CLAIM WORD, THIS CAUSES MSS
3584      / OF ANY TASK TO BE ACCEPTED.
3585      /ELSE DONT CLEAR TCB[0], KEEPING ON THE CLAIM.
3586      XCHRCQ,
3587 03303 1141      XWTMS, TAD      M5
3588 03304 1164      TAD      CURTSK
3589 03305 3150      DCA      MON0      /\PTR TO TCB[0] CLAIM WORD.
3590 03306 1173      TAD      Z1000 /KEEP OPTION SPECIFIED?
3591 03307 0160      AND      MONFUNC
3592 03310 7650      SNA      CLA      /
3593 03311 3550      DCA      MON0      /\NO, CLEAR CLAIM WORD.
3594 03312 4774      JMS      SMSQ      /\SEARCH MSQ FOR APPROPRIATE MS.
3595 03313 2000      2000
3596      /AC2000      /\NO SUITABLE MS IN Q. SET TASK TO WAIT,

```

```

3597 03314 7410 SKP
3598
3599
3600 /SET TASK TO WAIT,
3601 /PERHAPS A WAITCONDITION IN AC.
3602 03315 1173 XSTALL, TAD Z1000
3603 03316 4772' TSWAIT, JMS CURWT /\REMOVE TASK FROM RUNQ,
3604 03317 1133 TAD C200;TIMEOUT=200 /\
3605 0200
3606 03320 0160 AND MONFUNC /\TIMEOUT OPTION SPECIFIED?
3607 03321 7640 SZA CLA /\
3608 03322 4767' JMS TQIN /\INSERT TASK IN TIMEOUT Q,
3609 03323 6001 ION /\
3610 03324 1366 TAD (400 /SWAPOUT OPTION SPECIFIED?
3611 03325 0160 AND MONFUNC
3612 03326 7640 SZA CLA
3613 03327 4773' JMS RTNTSK /YES. RETURN PAGES CONTAINING TASKS CODE.
3614 03330 5765' JMP MONOUT
3615
3616 03331 3145 AVPT;(0;AVPT=.-2
3617 03332 3364
3618 3331
3619 03364 0000
3620 03365 0474
3621 03366 0400
3622 03367 2647
3623 03370 0617
3624 03371 3450
3625 03372 0545
3626 03373 4000
3627 03374 3400
3628 03375 0731
3629 03376 4063
3630 03377 2740
3631 3400

```

PAGE

```

3632      /SMSQ, SEARCH MESSAGE Q.
3633      /SEARCHES EITHER THE RCQ OR THE RPQ FOR MSS THAT ARE ALLOWED TO
3634      /REACTIVATE THE CURRENT TASK. IF THERE ARE, A MS IS PASSED TO THE
3635      /TSK AND DELETED FROM MSQ.
3636      /ELSE IF THE CHECK ONLY BIT WAS SET (B0 IN MONFUNC) WE CONTINUE
3637      /THE TASK WITH AC=0.
3638      /OTHERWISE WE RETURN FROM THIS SUBROUTINE IN ORDER TO SET THE
3639      /TASK TO WAIT.
3640
3641      /SEARCH RCQ: MON0=PTR TO TCB[0] CLAIM WORD.
3642      /SEARCH RPQ: MON0=PTR TO TCB[2] WAITED RP WORD.
3643      /MON0=PTR TO HEAD OF Q -1.
3644      /ARG1=2000 IF WTMS, 4000 IF WTRP.
3645      SMSQ, 0
3646      03400 0000      AC3777
3647      03401 7350
3648      03402 0007      AND      MONFL
3649      03403 3007      DCA      MONFL      /CLEAR TASKS LINK ANYHOW.
3650      03404 1150      TAD      MON0      /PTR HEAD OF Q -1
3651      03405 6002      IOF
3652      03406 3156      DCA      INT2      /\
3653      03407 1150      TAD      MON0
3654      03410 7001      IAC
3655      03411 7410      SKP
3656      03412 1551      SMSQLP, TAD      MON1      /\
3657      03413 3151      DCA      MON1      /\PTR TO PRECESSOR MS.
3658      03414 7140      CMA CLL      /\NOTE: 0 ENDS THE Q.
3659      03415 1551      TAD      MON1      /\
3660      03416 3154      DCA      INT0      /\PTR TO (NEXT MS)[0].
3661      03417 7420      SNL
3662      03420 5241      JMP      SMSQEX      /\
3663      03421 1600      TAD      SMSQ      /\FETCH WAIT CONDITIONS
3664      03422 4314      JMS      CHKMS      /\MS ALLOWED TO REACTIVATE TSK?
3665      03423 5212      JMP      SMSQLP      /\NO; KEEP SEARCHING.
3666
3667      /PASS MS TO TASK AND DELETE MS FROM Q.
3668      MSTOAC, ISZ      INTO      /\PTR TO MS[1] LINK WORD.
3669      03424 2154      TAD      INTO      /\GET SUCCESSOR.
3670      03425 1554      DCA      MON1      /\DELETE MS FROM Q.
3671      03426 3551      DCA      INTO      /\CLEAR LINKWORD
3672      03427 3554      AC4000
3673      03430 7330      AND
3674      03431 0600      AND      SMSQ      /\
3675      03432 1007      TAD      MONFL      /\SET TASKS L TO INDICATE
3676      03433 3007      DCA      MONFL      /\ SEND OR REPORT.
3677      03434 7001      IAC
3678      03435 1154      TAD      INTO      /\GET PTR TO MS[2]
3679      03436 3004      SETAC, DCA      MONAC      /\STORE IN TASKS AC.
3680      03437 6001      ION
3681      03440 5777      JMP      MONEX
3682
3683      /SEARCH FAILED.
3684      /IF CHECK ONLY BIT SET CONTINUE TASK WITH AC=0.
3685      /ELSE RETURN FROM SUBROUTINE.
3686      03441 6001      SMSQEX, ION      /\
3687      03442 1160      TAD      MONFUNC
3688      03443 7710      SPA CLA      /CHECK ONLY?

```

```

3687 03444 5236      JMP      SETAC  /YES. CONTINUE WITH AC=0.
3688 03445 1600      TAD      SMSQ
3689 03446 2200      ISZ      SMSQ  /NO. RETURN.
3690 03447 5600      JMP      SMSQ
3691
3692          3400  TSTTM=SMSQ
3693          /DETACH TCB FROM ENTRY IN STL.
3694          /PTR TO STL[TSK,0] IN AC.
3695 03450 0000  RNTCB, 0
3696 03451 3200      DCA      TSTTM  /TEMP
3697 03452 1600      TAD      TSTTM  /GET PTR TO TCB[5].
3698 03453 1376      TAD      (11
3699 03454 3010      DCA      X0      /PTR TO TCB[15]
3700 03455 1600      TAD      TSTTM  /PTR TO TCB[5]
3701 03456 7500      SMA
3702 03457 5264      JMP      RTNTC2  /ANY TCB ACTUALLY ATTACHED TO STL?
3703 03460 1141      TAD      M5      /NO. DONT TOUCH THE ENTRY.
3704 03461 4572      JMS      ZFN
3705 03462 4356      AVL20
3706          /BAD TRIC: FIRST DELETE TCB, AND THEN STILL GET SOME INFO OUT.
3707 03463 1410      TAD      X0      /GET CONTENTS OF TCB[15]; DISK ADDRESS.
3708 03464 3600  RTNTC2, DCA      TSTTM  /STORE IN STL.
3709 03465 5650      JMP      RNTCB
3710

```

```

3711 03466 1004 XFILTTCB,TAD MONAC
3712 03467 4775 JMS GETTCB /RETURNS POINTER TO TCB[5]
3713 03470 1142 TAD M6
3714 03471 3010 DCA X0
3715 03472 7240 CLA CMA
3716 03473 1161 TAD MARG1
3717 03474 3011 DCA X1
3718 03475 1141 TAD M5
3719 03476 3151 DCA MON1
3720 03477 1374 TAD (-17
3721 03500 3152 DCA MON2
3722 03501 4051 FILLP, JMS CDFUF
3723 03502 1411 TAD I X1
3724 03503 6201 CDF 0
3725 03504 3410 DCA I X0
3726 03505 2151 ISZ MON1
3727 03506 5311 JMP .+3
3728 03507 2011 ISZ X1
3729 03510 2010 ISZ X0
3730 03511 2152 ISZ MON2
3731 03512 5301 JMP FILLP
3732 03513 5236 JMP SETAC
3733
3734 /CHECK IF THIS MS CAN REACTIVATE THE TASK,
3735 /INT0=PTR TO MS[0]
3736 /IF MS IS RECEIVED, INT2 POINTS AT TCB[0], AC=2000
3737 /IF MS IS REPORTED, INT2 POINTS AT TCB[2], AC=4000.
3738 /SKIPS IF CHECK SUCCEEDS,
3739 03514 0000 CHKMS, 0 /\
3740 03515 7710 SPA CLA /\
3741 03516 5322 JMP CHKMS2 /\
3742 03517 7350 AC3777 /\SEND,
3743 03520 0554 AND I INT0 /\
3744 03521 7410 SKP /\
3745 03522 1154 CHKMS2, TAD INT0 /\REPORT
3746 03523 1556 TAD I INT2 /\COMPARE TO CLAIM WORD OR WAITED REP
3747 03524 7640 SZA CLA /\
3748 03525 1556 TAD I INT2 /\ANYTHING SPECIFIED?
3749 03526 7640 SZA CLA /\
3750 03527 5714 JMP I CHKMS /\
3751 03530 2314 ISZ CHKMS /\
3752 03531 7350 AC3777 /\
3753 03532 0554 AND I INT0 /\
3754 03533 7041 CIA /\
3755 03534 3556 DCA I INT2 /\COPY B1-11 OF TCB OF SENDER
3756 /INT0 TCB[0] OF RECEIVER IN ORDER
3757 /TO CLAIM,
3758 03535 5714 JMP I CHKMS /\
3759
3760 03536 3331 AVPT;(0;AVPT=.-2
3761 03537 3573
3762 3536
3763
3764 03573 0000
3765 03574 7761

```

/APP. A4. MC8.4. MONITOR CALLS AND RE PAL8-V9H 02/14/77 PAGE 47-1

3766	03575	4063	
3767	03576	0011	
3768	03577	0731	
3769		3600	PAGE

```

3770                /REQCDF FUNCTION.
3771                /REQUEST A DATAFLD TO BECOME ACCESSIBLE VIA THE QUICK VRCDF
3772                /ROUTINE. AC B6-11 IS THE REQUESTED FLD.
3773    03600    1377    XREQCDF,TAD        (11
3774    03601    1164                TAD        CURTSK
3775    03602    3150                DCA        MON0        /PTR TO TCB[14].
3776    03603    1550                TAD        MON0
3777    03604    0145                AND        C7700
3778    03605    3550                DCA        MON0        /OLD REQCDF OUT.
3779    03606    1004                TAD        MONAC
3780    03607    0130                AND        C77
3781    03610    1550                TAD        MON0
3782    03611    3550                DCA        MON0        /NEW REQCDF IN.
3783    03612    4051                JMS        CDFUF        /PATCH THE VRCDF ROUTINE IN USERS FIELD
3784    03613    5776                JMP        VRCDF3-1
3785
3786                /HELP ROUTINE FOR STOPPING AND RESUMING TSKS.
3787                /STSK# IN MONAC.
3788    03614    0000    HLPST,    0
3789    03615    1004                TAD        MONAC
3790    03616    4775                JMS        GETTCB
3791    03617    1135                TAD        M1        /PTR TO TCB[4], WAITWORD.
3792    03620    6002                IOF        /\
3793    03621    3155                DCA        INT1        /\
3794    03622    1555                TAD        INT1        /\
3795    03623    5614                JMP        HLPST        /\
3796
3797                /STOP A TSK.
3798                /STSK# IN MONAC.
3799    03624    7001    STP0,    IAC                /\REMOVE TSK FROM RUNQ AND SET UP
3800    03625    3157                DCA        INT3        /\STOPDBIT.
3801    03626    1155                TAD        INT1        /\
3802    03627    4774                JMS        OURUNQ        /\
3803                /FALL BACK INTO XSTOP AGAIN.
3804    03630    4214    XSTOP,    JMS        HLPST        /\RETURNS CONTENTS OF WAITWORD.
3805    03631    0373                AND        (7701        /\ANY WAITCONDITION?
3806    03632    7650                SNA CLA        /\
3807    03633    5224                JMP        STP0        /\NO; SO FIRST REMOVE TASK FROM RUNQ.
3808    03634    7344                ACM2        /\
3809    03635    0555                AND        INT1        /\
3810    03636    7001                IAC                /\SET UP STOPDBIT.
3811    03637    3555                DCA        INT1        /\
3812    03640    2155                ISZ        INT1        /\PTR TCB[5]
3813    03641    1555                TAD        INT1        /\GET PTR TO TIMEOUT NODE IF ANY.
3814    03642    3214                DCA        HLPST        /\
3815    03643    3614                DCA        HLPST        /\REMOVE TSK FROM TOQ (OR INNOCENT
3816    03644    1155                TAD        INT1        /\CLEAR)
3817    03645    3555                DCA        INT1        /\SET INNOCENT PTR.
3818    03646    5772                JMP        SETAC        /\
3819
3820                /RESUME TSK PREVIOUSLY STOPPPED.
3821                /MONAC =STSK#.
3822    03647    4214    XRESUME,JMS        HLPST        /\
3823    03650    0136                AND        C7776        /\REMOVE STOPDBIT
3824    03651    3555                DCA        INT1        /\

```

/APP. A4. MC8.4. MONITOR CALLS AND RE PAL8-V9H 02/14/77 PAGE 48-1

3825	03652	1555	TAD I	INT1	/\GET WAITWORD
3826	03653	0145	AND	C7700	/\ANY OTHER WAITCONDITIONS?
3827	03654	7650	SNA CLA	:	/\
3828	03655	4771'	JMS	INRUNQ	/\NO SO INSERT IN RUNQ.
3829	03656	5772'	JMP	SETAC	/\

```

3830          /***** CHANGING INTERRUPT SLOTS.
3831          /
3832          /THE FOLLOWING INSTRUCTIONS CHANGE INTSLOTS (SEE INTR SECTION);
3833          /DISINTR (DISABLE DEVICE INTERRUPT), CNINTR (CONNECT SBR TO
3834          /INTSLOT), CLINTR (ATTACH MS TO INTSLOT), FRINTR (SET INTSLOT
3835          /INTO DISCARD MODE). WE ADDED HERE ONE RELATED FUNCTION, SIMINTR,
3836          /TO START UP INTERRUPT DRIVEN SECTIONS.
3837          /
3838          /THE INTSLOT TO BE CHANGED IS ALWAYS DETERMINED BY ITS SKPIOT,
3839          /THE SKIPFLAG INSTRUCTION IT STARTS WITH. THIS SKPIOT IS PASSED
3840          /IN AC.
3841
3842          /SEARCH SKPCHAIN FOR APPROPRIATE INTSLOT. MONAC*SKPIOT.
3843          /AT RETURN X0 PTS AT SKPIOT+2 AND AC HOLDS CONTENTS OF SKPIOT+1.
3844          MCINTR, 0
3845          03657 0000          TAD          (SKPCHN=1
3846          03660 1370          MCILP, DCA          X0
3847          03661 3010          TAD I          X0          /GET SKPIOT.
3848          03662 1410          SNA          /O TERMINATES THE SKPCHN.
3849          03663 7450          SYSPLT          /WE HIT ON THE 0-INSTR AT THE END OF THE
3850          03664 4576          /SKPCHN, SO A NONEXISTING INTSLOT WAS
3851          /SPECIFIED.
3852          03665 7041          CIA
3853          03666 1004          TAD          MONAC
3854          03667 7650          SNA CLA
3855          03670 5274          JMP          MCILF          /WE FOUND THE INTSLOT.
3856          03671 7307          AC4
3857          03672 1010          TAD          X0
3858          03673 5261          JMP          MCILP          /NEXT INTSLOT.
3859          03674 6002          MCILF, IOF          /\CHANGING INTSLOTS MUST BE DONE DEAF OF
3860          03675 1410          TAD I          X0          /\COURSE
3861          03676 5657          JMP I          MCINTR /\
3862
3863
3864          /DISABLE DEVICE INTERRUPT.
3865          /CHANGE INTSLOT INTO
3866          /          SKPIOT
3867          /          JMP          .+4
3868          /          JMP          .+3
3869          /          ....
3870          /NOTE!!!!
3871          /          BE SURE THAT THE FLAG REMAINS CLEARED WHILE EXECUTING THE
3872          /          DISINTR CALL!
3873          03677 4257          XDISINTR,JMS          MCINTR /\
3874          03700 5312          JMP          MCIOUT /\
3875
3876
3877          /CONNECT A SBR TO INTSLOT.
3878          /SBR RESIDES IN TASKS DF. BE SURE THAT IT IS LOCKED!
3879          /INTSLOT IS CHANGED INTO
3880          /          SKPIOT
3881          /          JMP          .+4
3882          /          CIF CDF          /TO FLD OF SBR (CALLER'S DF).
3883          /          JMS I          .+1
3884          /          ENTRY          /PTR TO SBR ENTRYPOINT.

```

```

3885      /ARG1=PTR TO SBR.ENTRYPOINT.
3886      /ARG2=PTR TO MS[2] OF COMMUNICATION MS.
3887      03701 4325 XCNINTR,JMS STTOMS /STORE TCBPTR INTO COMM.MS, IF ANY.
3888      03702 4257 JMS MCINTR /\
3889      03703 1367 TAD (-1+4600-5200/\CHANGE "JMP .+4" INTO "JMS I .+3"
3890      03704 3257 DCA MCINTR /\TEMP
3891      03705 4350 JMS CDIFDF /\RETURNS CIP CDF TO DATAFIELD (LOCKED!).
3892      03706 3410 DCA X0 /\
3893      03707 1257 TAD MCINTR /\
3894      03710 3410 DCA X0 /\
3895      03711 1161 TAD MARG1 /\
3896      03712 3410 MCOUT, DCA X0 /\
3897      03713 5772 JMP SETAC /\
3898
3899
3900      /CLAIM AN INTSLOT.
3901      /ATTACH A MS TO INTSLOT.
3902      /THE MS WILL BE REPORTED TO THE CALLER EACH TIME THE SPECIFIED
3903      /DEVICE INTERRUPTS.
3904      /INTSLOT WILL BE CHANGED INTO:
3905      / SKPIOT
3906      / JMP .+4
3907      / CLRPIOT /CLEAR DEVICEFLAG
3908      / IEXIT /REPORT MS
3909      / MS /PTR TO MS[2].
3910      /ARG1=CLRPIOT (TO CLEAR DEVICE FLAG)
3911      /ARG2=PTR TO MS[2].
3912      03714 4325 XCLINTR,JMS STTOMS /STORE TCBPTR INTO SENDERWORD OF MS.
3913      03715 4257 XCL12, JMS MCINTR /\
3914      03716 7200 CLA /\
3915      03717 1161 TAD MARG1 /\
3916      03720 3410 DCA X0 /\
3917      03721 1366 TAD (IEXIT /\
3918      03722 3410 DCA X0 /\
3919      03723 1162 TAD MARG2 /\
3920      03724 5312 JMP MCOUT /\
3921
3922      03725 0000 STTOMS, 0 /STORE TCBPTR IN MS[0]
3923      03726 4765 JMS INARG2
3924      03727 1162 TAD MARG2 /
3925      03730 7450 SNA
3926      03731 5725 JMP STTOMS /NO MESSAGE INDICATED
3927      03732 1136 TAD M2
3928      03733 3150 DCA MON0 /PTR TO MS[0]
3929      03734 1164 TAD CURTSK /
3930      03735 3550 DCA MON0 /DENOTE IN SENDERWORD.
3931      03736 5725 JMP STTOMS
3932
3933
3934      /FREE AN INTSLOT.
3935      /SET INTO DISCARD MODE.
3936      /INTSLOT IS CHANGED INTO
3937      / SKPIOT
3938      / JMP .+4
3939      / CLRPIOT /CLEAR DEVICE FLAG

```

```

3940      /      IEXIT
3941      /      0
3942      /ARG1=CLRIOT.
3943 03737 3162 XFRINTR,DCA MARG2
3944 03740 5315      JMP      XCL12
3945
3946
3947      /START UP A DEAF INTERRUPTDRIVEN SECTION.
3948      /THE CALLER IS CONTINUED AS SOON AS THE NORMAL EXIT FROM
3949      /CONNECTED ROUTINE (IEXIT) IS TAKEN.
3950      /THE DEAF SECTION TO BE STARTED MUST RESIDE IN CALLERS DF.
3951      /ARG1=PTR TO ENTRYPOINT OF DEAF SECTION.
3952      /NOTE! SCED IS INHIBITED, SO THE DEAF SECTION CAN ONLY END BY
3953      /TAKING FSTEXIT!
3954 03741 6002 XSIMINTR,IOF      /-
3955 03742 1364      TAD      (EXTPRV) /-PATCH EXIT FROM INTR SECTION, SO THAT
IT
3956 03743 3763      DCA      EXTALL  /\JUMPS TO MONEX.
3957 03744 4350      JMS      CDIFDF  /-RETURNS CIF CDF TO DATAFIELD IN AC.
3958 03745 3346      DCA      .+1      /-
3959 03746 6203      CIF CDF      /-TO CALLER'S DATAFLD.
3960 03747 5561      JMP I      MARG1  /!
3961
3962      /COMPUTE CIF CDF TO DATAFIELD. (MUST BE LOCKED!)
3963 03750 0000      CDIFDF, 0
3964      IFDEF PDP8E <
3965 03751 1007      TAD      MONFL
3966 03752 0123      AND      C7
3967 03753 7106      CLL RTL;RAL
3968 03754 7004
3969 03755 1016      TAD      XCDIF
3970      >
3971      IFNDEF PDP8E <
3972      AC2
3973      TAD      CDFINS
3974      >
3975 03756 5750      JMP I      CDIFDF
3976
3977 03757 3536      AVPT;(0;AVPT=-2
3978 03760 3762
3979      3757
3980
3981 03762 0000
3982 03763 0422
3983 03764 5776
3984 03765 2740
3985 03766 4515
3986 03767 7377
3987 03770 0206
3988 03771 1076
3989 03772 3436
3990 03773 7701
3991 03774 1136
3992 03775 4063
3993 03776 0725
3994 03777 0011

```

/APP. A4. MC8,4. MONITOR CALLS AND RE PAL8-V9H 02/14/77 PAGE 49-3
3995 4000 PAGE

```

3996          /DISMISS CURRENT TSK FROM CORE.
3997          /THE PAGES CONTAINING TSKS CODE ARE RETURNED BUT THE TCB IS KEPT.
3998          /BASE:=CURTSK, MARG2:=PTR TO STL[TSK,1].
3999          /SET UP ONDISK CONDITION IN TCB.
4000          /USES MARG1.
4001 04000 0000 RTNTSK, 0
4002 04001 1164      TAD      CURTSK
4003 04002 3107      DCA      BASE
4004 04003 4060      GET;1
4005 04004 0001
4006 04005 0132      AND      C177
4007 04006 7124      STL RAL
4008 04007 1116      TAD      ZSTL
4009 04010 3162      DCA      MARG2 /PTR TO STL[TSK,1]
4010 04011 1562      TAD      MARG2 /LENGTH +ZREQ+CRES
4011 04012 0377      AND      (537
4012 04013 3161      DCA      MARG1 /ARGUMENT FOR PGRQN,
4013 04014 7330      AC4000 /TURN ON ONDISK BIT.
4014 04015 1510      TAD      X
4015 04016 3510      DCA      X
4016 04017 4060      GET;11 /CONTENTS OF TCB[14]; FIRST PAGE
4017 04020 0011
4018 04021 0146      AND      C7600
4019 04022 3110      DCA      X /TEMP
4020 04023 1110      TAD      X /SUBSTRACT PAGE# FROM PC.
4021 04024 7041      CIA      /FOR WE MUST SAVE IT RELATIVE.
4022 04025 1054      TAD      MONPC
4023 04026 3054      DCA      MONPC
4024 04027 1163      TAD      VCURIF /THE FLD IN WHICH CURTSK RESIDES
4025 04030 7002      BSWR
4026 04031 1110      TAD      X
4027 04032 4776      JMS      PGRQN
4028 04033 5600      JMP      RTNTSK
4029
4030
4031          /DETACH TCB FROM ENTRY IN STL.
4032          /STSK# IN MONAC.
4033 04034 1004 XRTNTCB, TAD      MONAC
4034 04035 0132      AND      C177 /COMPUTE PTR TO STL[TSK,0].
4035 04036 7104      CLL RAL
4036 04037 1116      TAD      ZSTL
4037 04040 4775      JMS      RNTCB
4038 04041 5774      JMP      SETAC
4039
4040
4041
4042          /REQUEST A NUMBER OF PAGES.
4043          /ARG1 B5=1: PAGES IN CORERESIDENT FLD, B7-11 LENGTH.
4044          /AT RETURN: FIRST PAGE IN AC B0-4, FLD IN AC B6-11.
4045
4046          /RETURN PAGES PREVIOUSLY REQUESTED BY REQPAQ.
4047          /ARG1 B5=1: CRES, B7-11 LENGTH.
4048          /AC B0-4: FIRST PAGE, B6-11: FLD.
4049 04042 3004 XREQPAQ,DCA      MONAC /JUST FOR SURE.
4050 04043 1161 XRTNPAG,TAD      MARG1

```

```

4051 04044 0132      AND      C177      /DONT BOTHER ABOUT BIT40.
4052 04045 3161      DCA      MARG1
4053 04046 1004      TAD      MONAC
4054 04047 4776'     JMS      PGRQN
4055 04050 5774'     JMP      SETAC
4056
4057                /REQUEST A FLD FOR DATA STORAGE,
4058                /AT RETURN: AC B6-11 =FLD.
4059 04051 1173      XREQFLD,TAD Z1000;FDATA=1000 /SET UP DATA STORAGE BIT IN FLDTAB.
4060                1000
4061 04052 4773'     JMS      GETFLD
4062 04053 3004      DCA      MONAC
4063 04054 1141      TAD      C7773;FMININ=4
4064                0004
4065 04055 0553      AND      MON3
4066 04056 3553      DCA      MON3      /CLEAR MONINBIT,
4067 04057 5772'     JMP      MONEX
4068
4069                /RETURN FLD PREVIOUSLY REQUESTED BY REQFLD.
4070                /AC B6-11 =FLD.
4071 04060 1004      XRTNFLD,TAD      MONAC
4072 04061 4771'     JMS      FRFLD
4073 04062 5774'     JMP      SETAC
4074

```

```

4075          /GET PTR TO TCB[5] TSK LINKWORD.
4076          /AC=STSK#.
4077          /USES BASE,X,X0.
4078 04063 0000 GETTCB, 0
4079 04064 0132 AND C177
4080 04065 7104 CLL RAL
4081 04066 1116 TAD ZSTL
4082 04067 3200 DCA RTNTSK /JUST A TEMP.
4083 04070 1600 TAD RTNTSK /FETCH STL[TSK,0]
4084 04071 7550 SPA SNA /IS A TCB ATTACHED TO THIS ENTRY?
4085 04072 5663 JMP GETTCB /YES, THEN THIS IS THE CORRECT PTR.
4086          IFDEF CHECK <
4087 04073 7650 SNA CLA
4088 04074 4576 SYSHLT /0 INDICATES FREE ENTRY IN STL, ERROR.
4089          >
4090 04075 4770 JMS GN20 /REQUEST SPACE FOR SETTING UP TCB
4091 04076 1010 TAD X0
4092 04077 3107 DCA BASE /PTR TO TCB[0]-1
4093 04100 1144 TAD M20
4094 04101 3110 DCA X /CTR.
4095 04102 3410 DCA X0 /CLEAR TCB
4096 04103 2110 ISZ X
4097 04104 5302 JMP .-2
4098 04105 1600 TAD RTNTSK /GET CONTENTS OF STL[TSK,0]
4099 04106 4075 PUT; 17+1 /STORE IN TCB[15].
4100 04107 0020
4101 04110 7327 AC6
4102 04111 1107 TAD BASE
4103 04112 3600 DCA RTNTSK /STORE PTR TO TCB[5] LINKWORD IN
4104 04113 2200 ISZ RTNTSK /STL[TSK,0]. PTR TO STL[TSK,1].
4105 04114 1600 TAD RTNTSK /GET PRIO
4106 04115 7002 BSWR
4107 04116 0127 AND C70
4108 04117 7132 STL RTR /2*PRIO+FWTMS
4109 04120 4075 PUT;4+1 /IN TCB[4].
4110 04121 0005
4111 04122 1200 TAD RTNTSK /PTR TO STL[TSK,1].
4112 04123 0134 AND C377 /COMPUTE STSK#
4113 04124 7130 STL RAR /STSK# +ONDISKBIT.
4114 04125 4075 PUT; 6+1 /IN TCB[6].
4115 04126 0007
4116 04127 7040 CMA
4117 04130 1110 TAD X /PTR TO TCB[5] LINKWORD
4118 04131 5663 JMP GETTCB
4119
4120          /REQUEST ENTRY IN STL.
4121          /ARG1,ARG2 CONTENTS OF THE ENTRY.
4122 04132 4767 XREQSTL,JMS INARG2
4123 04133 1116 TAD ZSTL
4124 04134 3110 DCA X /PTR IN STL.
4125 04135 1366 TAD (=MAXSTL
4126 04136 3200 DCA RTNTSK /CTR
4127 04137 1510 RQSTLP, TAD X
4128 04140 7650 SNA CLA
4129 04141 5347 JMP RQSTLF /FOUND.

```

4130	04142	2110	ISZ	X	
4131	04143	2110	ISZ	X	
4132	04144	2200	ISZ	RTNTSK	
4133	04145	5337	JMP	RQSTLP	/NEXT.
4134	04146	4576	SYSHLT		/ALL ENTRIES IN STL USED.
4135	04147	1161	RQSTLF, TAD	MARG1	
4136	04150	3510	DCA I	X	
4137	04151	2110	ISZ	X	
4138	04152	1162	TAD	MARG2	
4139	04153	3510	DCA I	X	
4140	04154	1110	TAD	X	
4141	04155	7010	RAR		
4142	04156	0132	AND	C177	
4143	04157	5774	JMP	SETAC	
4144					
4145			/ATTACH TCB TO ENTRY IN STL.		
4146	04160	1004	XREQTCB, TAD	MONAC	/GET STSK#.
4147	04161	4263	JMS	GETTCB	
4148	04162	5774	JMP	SETAC	/RETURN PTR TO TCB[5]
4149					
4150	04163	3757	AVPT;(0;AVPT=-2		
4151	04164	4165			
4152		4163			
4153					
4154	04165	0000			
4155	04166	7701			
4156	04167	2740			
4157	04170	4354			
4158	04171	1507			
4159	04172	0731			
4160	04173	1447			
4161	04174	3436			
4162	04175	3450			
4163	04176	1221			
4164	04177	0537			
4165		4200	PAGE		

```

4166          /***** LOCKING AND UNLOCKING FIELDS *****/
4167          /FUNCTIONS: LOCK AND UNLOCK
4168          /THE FIELD TO BE LOCKED OR UNLOCKED IS ALWAYS PASSED IN DF
4169          /(TASKS DATAFIELD).
4170
4171          04200 1146 XUNLOCK,TAD      M200
4172          04201 1131 XLOCK, TAD      C100
4173          /CHANGE IN LOCK COUNT IN AC.
4174          /AC=100: LOCK, AC=-100: UNLOCK TASKS DF.
4175          04202 3150          DCA      MON0      /TEMP.
4176          04203 1007          TAD      MONFL     /GET DF
4177          04204 0123          AND      C7
4178          04205 1170          TAD      ZCORMAP
4179          04206 3151          DCA      MON1      /PTR IN CORMAP.
4180          04207 1551          TAD I    MON1
4181          04210 1150          TAD      MON0
4182          IFDEF CHECK <
4183          04211 7510          SPA
4184          04212 4576          SYSHLT
4185          >
4186          04213 3551          DCA I    MON1
4187          04214 5777          JMP      MONEX
4188          /NOTE: THIS LOCK IS TEMPORARY, SO WE NEED
4189          /NOT SET THE CRESBIT IN FLDTAB.
4190

```

```

4191      /APP A5. MC8.5. TABLES, STORAGE ALLOCATOR.
4192      /***** TABLES AND STORAGE ALLOCATION *****/
4193
4194      /GET NODE.
4195      /      JMS      GN
4196      /AVLN,      0      /HEAD OF AVAIL LIST.
4197      /      NN      /LENGTH OF NODE, NEEDED IF NEW NODES ARE
4198      /      /CREATED.
4199      /IFDEF STATX <AVLNCT, 0 > /REQUESTED NODES COUNTER
4200      /NORMAL,...      /X0= (W0 OF NODE)-1.
4201
4202      04215  0000  GN,      0
4203      04216  1615      TAD      GN      /GET FIRST NODE
4204      04217  7450      SNA
4205      04220  5233      JMP      GNA      /THIS CHAIN EXHAUSTED; CREATE NEW NODE.
4206      04221  3362      DCA      GNFTM
4207      04222  1762      TAD      GNFTM
4208      04223  3615      DCA      GN      /UPDATE AVAIL LIST.
4209      04224  2215      ISZ      GN
4210      04225  2215      IFDEF STATX < ISZ GN >
4211      04226  2215  GNEX,  ISZ      GN
4212      04227  7040      CMA
4213      04230  1362      TAD      GNFTM
4214      04231  3010      DCA      X0      /SET AUTOINDEX TO NODE.
4215      04232  5615      JMP      GN
4216      /BREAK A NEW NODE FROM THE LARGE AVAILLIST IN CORE.
4217      04233  2215  GNA,      ISZ      GN      /GN PTS TO LENGTH NEEDED.
4218      04234  1376      TAD      (AVHEAD /SET PTR TO PRECESSOR,
4219      04235  3317  GNALP, DCA      GNPR      /SEARCH FOR NODE OF SUFFICIENT LENGTH.
4220      04236  1717      TAD      GNPR      /GET NODE
4221      04237  7450      SNA
4222      04240  4576      SYSHLT      /ALL AVAIL SPACE EXHAUSTED.
4223      04241  3362      DCA      GNFTM      /PTR TO W0 OF NODE
4224      04242  1362      TAD      GNFTM
4225      04243  3010      DCA      X0      /PTR TO PTR TO LAST LOC (AUTO IND).
4226      04244  1410      TAD      X0
4227      04245  3010      DCA      X0      /PTR TO LAST LOC.
4228      04246  1010      TAD      X0      /((LAST LOC)
4229      04247  7160      CMA STL      /L=1, AC=-(LAST LOC) -1
4230      04250  1362      TAD      GNFTM      /L=1; -LENGTH IN AC.
4231      04251  1615      TAD      GN      /ADD REQUESTED LENGTH
4232      /OK IF AC=0, OR NO OVERFLOW (L=1).
4233      /-LENGTH OF REMAINDER IN AC.
4234      04252  7460      SZA SNL
4235      04253  5300      JMP      GNNEXT      /NOT LONG ENOUGH.
4236      04254  7001      IAC      /L=0 IF REMAINDER <2.
4237      IFDEF STATX < /COUNT LOST SPACE
4238      04255  7450      SNA      /IF AC=0, LENGTH OF REMAINDER=1,
4239      04256  2304      ISZ      AVDIRT      /SO WE ARE GOING TO LOOSE 1 LOC.
4240      >
4241      04257  7620      SNL CLA      /IF (LENGTH REMAINDER)<2 (L=0),
4242      04260  5276      JMP      GNDEL      /THEN DELETE REMAINDER FROM AVAILLIST,
4243      GNBR,      /ELSE BREAK THE NODE.
4244      04261  1362      TAD      GNFTM      /PTR TO FIRST LOC OF NODE
4245      04262  1615      TAD      GN      /ADD LENGTH

```

4246	04263	3717	DCA	GNPR	/UPDATE PRECESSOR.
4247	04264	1717	TAD	GNPR	
4248	04265	3317	DCA	GNPR	/PTR TO W0 OF REMAINDER NODE.
4249	04266	1762	TAD	GNFTEM	/PTR TO NEXT NODE
4250	04267	3717	DCA	GNPR	
4251	04270	2317	ISZ	GNPR	
4252	04271	1010	TAD	X0	/STORE THERE NODE'S LAST LOC.
4253	04272	3717	GNEX2, DCA	GNPR	
4254			IFDEF STATX	<	
4255	04273	2215	ISZ	GN	
4256	04274	2615	ISZ	GN	/COUNT THE CREATED NODES
4257			>		
4258	04275	5226	JMP	GNEX	
4259			GNDEL,		/DELETE REMAINDER NODE FROM AVAILLIST,
4260	04276	1762	TAD	GNFTEM	/GET SUCCESSOR
4261	04277	5272	JMP	GNEX2	/STORE IN PRECESSOR.
4262	04300	7200	GNNEXT, CLA		/NODE TOO SHORT, TRY NEXT ONE
4263	04301	1362	TAD	GNFTEM	
4264	04302	5235	JMP	GNALP	
4265					
4266	04303	0000	AVHEAD, 0		
4267	04304	0000	IFDEF STATX	<AVDIRT, 0>	
4268		4317	GNPR=DF3		
4269					
4270					
4271			/	TAD	PTR
4272			/	JMS	FN
4273			/		AVLN
4274					/PTR TO AVAIL LIST.
4275	04305	0000	FN,	0	
4276	04306	3362	DCA	GNFTEM	/PTR TO NODE W0.
4277	04307	1705	TAD	FN	
4278	04310	3215	DCA	GN	/PTR TO AVAIL LIST.
4279	04311	1615	TAD	GN	
4280	04312	3762	DCA	GNFTEM	
4281	04313	1362	TAD	GNFTEM	
4282	04314	3615	DCA	GN	/INSERTED IN AVAIL LIST
4283	04315	2305	ISZ	FN	
4284	04316	5705	JMP	FN	
4285					
4286			/DELETE AND FREE A NODE FROM THE FCCHAIN.		
4287			/PGROLD PTS TO PRECESSOR.		
4288					
4289			GNPR,		
4290	04317	0000	DF3,	0	
4291	04320	1020	TAD	PGROLD	/PTR TO PRECESSOR
4292	04321	3362	DCA	GNFTEM	
4293	04322	1762	TAD	GNFTEM	
4294	04323	3215	DCA	GN	/PTR TO NODE TO BE DELETED.
4295	04324	1615	TAD	GN	
4296	04325	3762	DCA	GNFTEM	/DELETED FROM FCCHAN.
4297	04326	1215	TAD	GN	
4298	04327	4305	JMS	FN	
4299	04330	4342		AVL3	
4300	04331	5717	JMP	DF3	

```

4301
4302 04332 0000 GN2, 0
4303 04333 4215 JMS GN
4304 04334 0000 AVL2, 0
4305 04335 0002 2
4306 04336 0000 IFDEF STATX <AV2CT, 0 >
4307 04337 5732 JMP I GN2
4308
4309 04340 0000 GN3, 0
4310 04341 4215 JMS GN
4311 04342 0000 AVL3, 0
4312 04343 0003 3
4313 04344 0000 IFDEF STATX <AV3CT, 0 >
4314 04345 5740 JMP I GN3
4315
4316 04346 0000 GN5, 0
4317 04347 4215 JMS GN
4318 04350 0000 AVL5, 0
4319 04351 0005 5
4320 04352 0000 IFDEF STATX <AV5CT, 0 >
4321 04353 5746 JMP I GN5
4322
4323 04354 0000 GN20, 0
4324 04355 4215 JMS GN
4325 04356 0000 AVL20, 0
4326 04357 0020 20
4327 04360 0001 IFDEF STATX <AV20CT, 1 /TCB FOR MTINIT >
4328 04361 5754 JMP I GN20
4329
4330 04362 0000 GNFTM, 0
4331
4332 04363 4163 AVPT;(0;AVPT=-2
4333 04364 4375
4334 4363
4335
4336 04375 0000
4337 04376 4303
4338 04377 0731
4339 4400 PAGE

```

```

4340          /FORCE STATIC LIST AT MULTIPLE OF 400.
4341          IFNZRO .^200 < /ONE EXTRA PAGE
4342              AVPT;(0;AVPT=.-2
4343              PAGE
4344          >
4345
4346          /STATIC TASK LIST.
4347          STLST,
4348          0000 SIDLE=.#2^177
4349          04400 5113 IDLE;-1 /IDLELOOP TASK.
4350          04401 7777
4351          0001 SMTSWAP=.#2^177
4352          04402 5035 MTSWAP;-1 /MONITOR SWAP TASK.
4353          04403 7777
4354          0002 SMTFTCH=.#2^177
4355          04404 5051 MTFTCH;-1 /MONITOR FETCH TASK, PRIO=1!
4356          04405 7777
4357          0003 SMTTIME=.#2^177
4358          04406 5067 MTTIME;-1 /MONITOR TIMER TASK
4359          04407 7777
4360          0004 SMTSDSK=.#2^177
4361          04410 5103 MTSDSK;-1 /MONITOR SYSTEM DISK TASK
4362          04411 7777
4363          0005 SMTINIT=.#2^177
4364          04412 5127 MTINIT;1537 /INITIALIZATION TASK.
4365          04413 1537
4366          04414 0000 ZBLOCK MAXSTL+2+2*STLST=, /CLEAR REMAINING ENTRIES.
4367
4368          IFNZRO .-1^177-175^4000 <AVPT;(0;AVPT=.-2 > /SPACE LEFT ON THIS
4369                      /PAGE.
4370
4371          IFNZRO .^177 <PAGE>
4372

```

```

4373
4374 /FIELDTABLE
4375 IFNZRO :A177 <FLDTAB MUST BE ON PAGE BEGIN!>
4376 FLDTAB,
4377 04600 6707 6200+FMONIN+FOCCUP+FDP+FCRES+FZREQ
4378 04601 4220 4220
4379 04602 4230 4230
4380 04603 4240 4240
4381 04604 4250 4250
4382 04605 4260 4260
4383 04606 4270 4270
4384 04607 *FLDTAB+ACTMAX
4385 04607 0000 ZBLOCK VIRMAX=ACTMAX
4386 04677 *FLDTAB+VIRMAX
4387 04677 6713 6210+FZREQ+FCRES+FOCCUP+FDP /USED BY INIT TASK,
4388
4389 /PRIORITY RUNGS.
4390 RQPTRS,
4391 04700 0000 0;-1 /PRIO 0
4392 04701 4700 MTINIT; MTINIT /PRIO 1, INIT TASK IN.
4393 04702 5127
4394 04703 5127
4395 04704 0000 0;-1 /PRIO 2
4396 04705 4704
4397 04706 0000 0;-1 /PRIO 3
4398 04707 4706
4399 04710 0000 0;-1 /PRIO 4
4400 04711 4710
4401 04712 0000 0;-1 /PRIO 5
4402 04713 4712
4403 04714 0000 0;-1 /PRIO 6
4404 04715 4714
4405 04716 0000 0;-1 /PRIO 7
4406 04717 4716
4407 04720 5113 IDLE /PRIO 10. NO TAIL WORD NEEDED,
4408
4409 04721 0000 FCCHAN, 0 /HEAD OF FREE CORE CHAIN.
4410
4411 /COREMAP.
4412 CORMAP,
4413 04722 0100 100 /FLD 0 LOCKED IN CORE,
4414 04723 0177 VIRMAX+100 /THE HIGHEST VIRTUAL FLD INITIALLY LOCKED
4415 / IN FLD 1.
4416 04724 0001 1;2;3;4;5;6
4417 04725 0002
4418 04726 0003
4419 04727 0004
4420 04730 0005
4421 04731 0006
4422
4423 4732 *CORMAP+ACTMAX+1
4424
4425 04732 0000 IFDEF STATX <MONSTAT, ZBLOCK 100 /DOUBLE PRECISION COUNTERS OF
4426 /MONCALLS>
4427

```

```

4428          IFZERO .^4000 <*4000 /FORCE TCB'S ABOVE 4000>
4429          /TCB OF MONITOR SWAP TASK.
4430          /      0;0          /CLAIMWORD, HEAD OF RCQ; UNUSED,
4431      05032  5743          -SWAPMS+2          /WAITED RP.
4432      05033  0000          0          /HEAD OF RPQ.
4433      05034  4000          0+2+FWTRP          /WAITS +PRIO
4434      05035  5035      MTSWAP, .          /INNOCENT PTR IN LINKWORD
4435      05036  0001          SMTSWAP          /STSK#.
4436      05037  0000          0          /FLDS
4437      05040  0000          0          /AC
4438      05041  2042          MTSWIN          /PC
4439      05042  0000          0          /MQ; UNUSED.
4440      05043  0000          0          /EAE+L
4441          /      0;0;0;0          /BASE, X, FIRST PAGE, DISKADDR; UNUSED,
4442
4443          /TCB OF MONITOR FETCH TASK.
4444      05044  0000          0;0          /CLAIMWORD, HEAD OF RCQ,
4445      05045  0000
4446      05046  0000          0;0          /WAITED RP, HEAD OF RPQ.
4447      05047  0000
4448      05050  2002          1+2+FWTMS          /WAITS +PRIO
4449      05051  5051      MTFTCH, .          /LINK WORD.
4450      05052  0002          SMTFTCH          /STSK#.
4451      05053  0000          0          /FLDS
4452      05054  0000          0          /AC
4453      05055  2402          MTFTIN+2          /PC
4454      05056  0000          0          /MQ
4455      05057  0400          400          /EAE+L, RUNS WITH SCDINH.
4456      05060  0000          0;0;0;0          /BASE, X, FIRST PAGE, DISKADDR,
4457      05061  0000
4458      05062  0000
4459      05063  0000
4460
4461          /TCB OF MONITOR TIMER TASK.
4462          /      0;0          /CLAIMWORD, HEAD OF RCQ; UNUSED,
4463      05064  5114          -TIMEMS+2          /WAITED RP
4464      05065  0000          0          /HEAD OF RPQ,
4465      05066  4000          0+2+FWTRP          /WAITS +PRIO
4466      05067  5067      MTTIME, .          /LINK WORD.
4467      05070  0003          SMTTIME          /STSK#.
4468      05071  0000          0          /FLDS
4469      05072  0000          0          /AC
4470      05073  2600          TIMEIN          /PC
4471      05074  0000          0          /MQ
4472      05075  0000          0          /EAE+L
4473          /      0;0;0;0          /BASE, X, FIRST PAGE, DISK ADDR; UNUSED,
4474
4475          /TCB OF MONITOR SYSTEM DISK TASK.
4476      05076  0000          0;0          /CLAIMWORD, HEAD OF RCQ,
4477      05077  0000
4478      05100  0000          0;0          /WAITED RP, HEAD OF RPQ.
4479      05101  0000
4480      05102  2000          0+2+FWTMS          /WAITS +PRIO
4481      05103  5103      MTSDSK, .          /LINK WORD.
4482      05104  0004          SMTSDSK          /STSK#

```

```

4483 05105 0000 0 /FLDS
4484 05106 0000 0 /AC
4485 05107 2103 SDIN+2 /PC
4486 05110 0000 0 /MQ
4487 05111 0000 0 /EAE+L
4488 / 0;0;0;0 /BASE,X,FIRST PAGE, DISK ADDR; UNUSED.
4489
4490 /TCB OF IDLE LOOP TASK.
4491 / 0;0 /CLAIMWORD, HEAD OF RCQ; UNUSED.
4492 / 0;0 /WAITED RP, HEAD OF RPQ; UNUSED.
4493 05112 0020 1042 /WAITS +PRIO
4494 05113 0000 IDLE, 0 /LINK WORD
4495 05114 0000 SIDLE /STSK#.
4496 05115 0000 0 /FLDS
4497 05116 0000 0 /AC
4498 05117 2352 IDLEIN /PC
4499 05120 0000 0 /MQ
4500 05121 0000 0 /EAE+L
4501 / 0;0;0;0 /BASE,X,FIRST PAGE,DISK ADDR; UNUSED.
4502
4503 /TCB OF INIT TSK.
4504 05122 0000 0;0 /CLAIMWORD, HEAD OF RCQ,
4505 05123 0000 0;0 /WAITED RP, HEAD OF RPQ,
4506 05124 0000 0;0 /WAITED RP, HEAD OF RPQ,
4507 05125 0000 2 /PRIO 1.
4508 05126 0002 MTINIT, 0 /LINKWORD, LAST IN ITS Q.
4509 05127 0000 SMTINIT /STSK#.
4510 05130 0005 VIRMAX+100+VIRMAX/FLDS
4511 05131 7777 0 /AC
4512 05132 0000 INITIN /PC
4513 05133 2000 INITIN=2000
4514 2000 /MQ
4515 05134 0000 0 /EAE+L
4516 05135 0000 0;0 /BASE AND X.
4517 05136 0000 200 /FIRSTPAGE +REQCDF
4518 05137 0000 0 /DUMMY DISK ADDRESS. WILL CLEAR THE ENTRY IN STL
4519 05140 0200 /ON EXIT.
4520 05141 0000
4521
4522
4523 05142 4363 AVPT;7377;AVPT=-2/LOTS OF AVAIL SPACE.
4524 05143 7377
4525 5142
4526
4527
4528 /NOTE: WE RESERVED TWO PAGES FOR SIMPLE ERRORHANDLING
4529 /AND RESTART OF OS/8. IF THESE PAGES ARE ALSO TO BE USED
4530 /IN THE AVAILLIST SYSTEM, THEN BE AWARE OF (WORDCOUNT-
4531 /CORE ADDRESS) LOCATIONS OF SOME DATABREAK DEVICES.
4532 /PREFERABLY DONT USE LOC 7750-7755!

```

\$

PAL8-V9H 02/14/77 PAGE 56

4533

\$

\$

PAL8-V9H 02/14/77 PAGE 57

ACTMAX	0007	C7777	0135	GN2	4332	KRB3	6466
AVDIRT	4304	DCDIR	0200	GN20	4354	KRB1	6034
AVHEAD	4303	DFFARM	0100	GN3	4340	KRS2	6424
AVL2	4334	DF3	4317	GN5	4346	KRS3	6464
AVL20	4356	DISINT	0016	GPZCDF	1614	KSF1	6031
AVL3	4342	DK8EP	0004	GPZCT	1634	KSF2	6421
AVL5	4350	EAE	0001	GPZRIN	1600	KSF3	6461
AVPT	5142	ERHLT	4023	GPZR12	1632	LOCK	0046
AV2CT	4336	EXIT	0054	GPZTAD	1625	MARG1	0161
AV20CT	4360	EXTALL	0422	HARDMG	0001	MARG2	0162
AV3CT	4344	EXTPRV	5776	HCT	0370	MAXDEV	0013
AV5CT	4352	FCCHAN	4721	HLPST	3614	MAXSTL	0077
BASE	0107	FCHECK	2000	HLTINS	0003	MCILF	3674
BSWR	7002	FCHEX	0744	HP	0430	MCILP	3661
CALL	4054	FCRES	0100	HPRIO	0165	MCINTR	3657
CCDF	0034	FCSWAP	2010	IDLE	5113	MCIOU	3712
CDFCUR	4051	FDATA	1000	IDLEIN	2352	MONAC	0004
CDFUF	0051	FDCMAX	5000	IDL	2350	MONEX	0731
CDIFDF	3750	FDP	0002	IDL2	2363	MONEX0	3261
CDIFI	0721	FILLP	3501	IDNS	2342	MONFL	0007
CHECK	0001	FILTCB	0065	IDNSHL	2341	MONFUN	0160
CHKMS	3514	FLDTAB	4600	IDPTR	2347	MONIN	0004
CHKMS2	3522	FLDTEM	1524	IDSW	2345	MONITO	3000
CHKONL	4000	FMININ	0004	IDSWHL	2344	MONIT2	3055
CHRCQ	4014	FMONIN	0004	LEXIT	4515	MONJMP	3103
CHRPQ	4005	FN	4305	INARG2	2740	MONOUT	0474
CLINTR	0023	FOCCUP	0001	INIT	0200	MONPC	0054
CMNSAF	0500	FRFLD	1507	INITIN	2000	MONSAF	0500
CNINTR	0021	FRINTR	0025	INRGEX	1127	MONSCD	0543
CORMAP	4722	FSTEXT	0417	INRUNQ	1076	MONSF2	0510
CUR	0000	FSTXT2	0445	INS200	6004	MONSTA	4732
CURTSK	0164	FTCHMS	2534	INTAC	0005	MON0	0150
CURWT	0545	FWTMS	2000	INTCT	0301	MON1	0151
C10	0124	FWTRP	4000	INTDEF	0000	MON2	0152
C100	0131	FWTSPW	0400	INTFL	0006	MON3	0153
C177	0132	FWTTM	0200	INTPC	0000	MON76	3072
C2	0120	FZREQ	0400	INTR	0177	MSAPP	1051
C200	0133	GET	4060	INTR2	0202	MSAPP1	1061
C3	0121	GETFLD	1447	INTSCD	0400	MSAPP2	1071
C37	0125	GETFLF	1466	INT0	0154	MSFREE	4014
C377	0134	GETFLP	1456	INT1	0155	MSIN	1023
C4	0122	GETFLX	1504	INT2	0156	MSNOD1	2717
C40	0126	GETFL2	1503	INT3	0157	MSNOD2	2735
C7	0123	GETMQ	7701	ITSAF	0464	MSNREQ	2730
C70	0127	GETTCB	4063	JUMS	4070	MSQMAX	1075
C7600	0146	GN	4215	KB1	0006	MSREQ	4014
C77	0130	GNA	4233	KB2	0010	MSTOAC	3424
C7700	0145	GNALP	4235	KB3	0012	MTFCT	2531
C7760	0144	GNBR	4261	KCC1	6032	MTFFLD	2455
C7770	0143	GNDEL	4276	KCC2	6422	MTFPAG	2535
C7772	0142	GNEX	4226	KCC3	6462	MTFTCH	5051
C7773	0141	GNEX2	4272	KEEP	1000	MTFTIN	2400
C7774	0140	GNFTEM	4362	KM8E	0001	MTINIT	5127
C7775	0137	GNNEXT	4300	KRB1	6036	MTSDSK	5103
C7776	0136	GNPR	4317	KRB2	6426	MTSWAP	5035

\$

MTSWEX 2063	REQCDF 0052	SDSWP 2130	TSF1 6041
MTSWIN 2042	REQFLD 0034	SDSW1 2122	TSF2 6431
MTTIME 5067	REQPAG 0031	SETAC 3436	TSF3 6471
MYCDIF 0052	REQSTL 0041	SIDLE 0000	TSKSAF 0554
MYCIF 0021	REQTCB 0042	SIMINT 0027	TSKSWT 0452
M1 0135	RESUME 0060	SKPCHN 0207	TSTTM 3400
M10 0143	RKBLK 2327	SKPEND 0301	TSWAIT 3316
M100 0145	RKCHK 2325	SKPFIT 0375	TT1 0005
M2 0136	RKCMD 2322	SMSQ 3400	TT2 0007
M20 0144	RKDONE 2315	SMSQEX 3441	TT3 0011
M200 0146	RKERCT 2321	SMSQLP 3412	UNLOCK 0050
M3 0137	RKERR 2312	SMTFTC 0002	VCALL 0054
M4 0140	RKER2 2217	SMTINI 0005	VCDF 4025
M5 0141	RKER3 2235	SMTSDS 0004	VCDFTM 0042
M6 0142	RKINTR 2333	SMTSWA 0001	VCDF1 3032
M7771 0123	RKLP 2260	SMTTIM 0003	VCDIF 0051
M7774 0122	RKMS 2325	SNDMS 0003	VCURIF 0163
M7775 0121	RKMSP 2213	SNDREP 1000	VERHLT 0023
M7776 0120	RKPAG 2326	SNDWTR 0007	VFBLOK 4000
NONFIT 0000	RKRTRY 2240	STALL 0200	VFLCT 1600
NONREP 2000	RK600 2245	STATX 0001	VFLESS 1657
NRT 0400	RK8E 0000	STLST 4400	VFLF 1700
ONDISK 0600	RLOCLP 2471	STOP 0056	VFLOCK 1427
OURQF 1160	RLOC1 2501	STOPD 0001	VFLTEM 1445
OURQLP 1150	RLOC2 2507	STORMQ 7421	VFLTM2 1446
OURUNQ 1136	RNTCB 3450	STP0 3624	VFLTRY 1662
PDP8E 0001	RP 0013	STTOMS 3725	VFNEXT 1656
PDP8IE 0002	RPINTR 0402	SWAPMS 2037	VFSWLP 2042
PGR 1200	RQPTRS 4700	SWNODE 2150	VGET 0060
PGREX 1261	RQSTLF 4147	SWPOUT 0400	VIRMAX 0077
PGREXH 1215	RQSTLP 4137	SYDISK 0000	VJUMS 0070
PGRFF 1351	RTN 2705	SYSHLT 4576	VMSNOD 0014
PGRLEN 0025	RTNFLD 0036	SYTIME 0004	VPUT 0075
PGRLP 1210	RTNPAG 0033	SYWAIT 2330	VRCDF 4020
PGRLPM 1213	RTNTCB 0044	TCF1 6042	VRCDF1 3042
PGRMB 1327	RTNTC2 3464	TCF2 6432	VRCDF3 0726
PGRMF 1315	RTNTSK 4000	TCF3 6472	VRDF 4045
PGRM2 1321	SCDF 0633	TCKLEN 6331	VRDF1 2671
PGRNM 1332	SCDF2 0716	TIME 2667	VVCDF 0025
PGROLD 0020	SCDF3 0730	TIMEIN 2600	VVRCDF 0020
PGRQ 1224	SCDINH 0167	TIMEMS 2666	VVRDF 0045
PGRQLP 1227	SCDLP2 0625	TIMEOU 0200	WTMS 0014
PGRQN 1221	SCDREQ 0166	TLS1 6046	WTRP 0005
PGRQX 1400	SCED 0617	TLS2 6436	WTRPMT 2644
PGRREM 0045	SCEDLP 0622	TLS3 6476	X 0110
PGRTEM 0070	SCEDNI 0754	TOQEX 2642	XCDF70 0117
PGRTN 1266	SDERCT 2201	TOGFN 2630	XCDIF 0016
PGRTN2 1303	SDEX 2076	TOQLP 2607	XCHRCQ 3303
PTBASE 0654	SDFCT 2155	TOQLP2 2640	XCHRPQ 3236
PTX 0701	SDGO 2202	TPC1 6044	XCLINT 3714
PUT 4075	SDIN 2101	TPC2 6434	XCL12 3715
POCEX 1645	SDMSP 2100	TPC3 6474	XCNINT 3701
POCLST 1647	SDNCT 2153	TQHEAD 2663	XDISIN 3677
POCTAD 1634	SDNORM 2067	TQIN 2647	XERROR 3144
RDFLD 1355	SDPTR 2200	TSBLOK 6000	XEXIT 3267

\$

PAL8-V9H 02/14/77 PAGE 57-2

XFILTC	3466
XFLDTA	0112
XFRINT	3737
XLOCK	4201
XMONIT	0111
XMSNOD	0114
XREQCD	3600
XREQFL	4051
XREQPA	4042
XREQST	4132
XREQTC	4160
XRESUM	3647
XRP	3251
XRP2	3263
XRTNFL	4060
XRTNPA	4043
XRTNTC	4034
XSIMIN	3741
XSNDMS	3200
XSNDWT	3200
XSND2	3212
XSTALL	3315
XSTOP	3630
XUNLOC	4200
XVCALL	0037
XVRDF1	0113
XWTMS	3303
XWTRP	3236
X0	0010
X1	0011
X2	0012
X3	0013
X6000	0001
ZCORMA	0170
ZFN	0172
ZSNDRE	0171
ZSTL	0116
Z1000	0173

ERRORS DETECTED: 0
LINKS GENERATED: 81
4534

ACM2	910	1441	2511	2997	3510	3548	3808				
ACM3	2481	3531									
ACS	1260										
ACTMAX	14#	2139	2142	4384	4385	4423					
ACTUAL	13										
AC2	1468	2090	2426	2500	2608	2616	3972				
AC2000	1398	3513									
AC3	1472	2911									
AC3777	1200	2273	2417	2762	3646	3742	3752				
AC4	1456	2955	3522	3856							
AC4000	1475	2410	2900	3671	4013						
AC6	2503	4101									
ALLOCA	3381										
ANALEX	740	741									
ASS	99	256	446	505	547						
AVDIRT	4239	4267#									
AVHEAD	4218	4266#									
AVL2	2894	4304#									
AVL20	3705	4325#									
AVL3	4299	4311#									
AVL5	3000	3560	4318#								
AVPT	545	883	885	1172	1174	1405	1407	1608	1610	1883	
	1885	2033	2035	2160	2162	2451	2454	2703	2709	2828	
	2830	3022	3024	3386	3388	3616	3618	3760	3762	3977	
	3979	4150	4152	4332	4334	4342	4368	4523	4525#		
AV2CT	4306#										
AV20CT	4327#										
AV3CT	4313#										
AV5CT	4320#										
BASE	399	438	449#	1143	1228	1229	1241	1306	2723	2813	
	4003	4092	4102								
BE	99	256	4375								
BEGIN	4375										
BSWR	242	246#	1244	1309	1871	1876	2735	2752	4025	4106	
CALL	377#	2394	2397	2483	2486	2638	2720	2768			
CCDF	336#	3323									
CDFCUR	362#										
CDFINS	1304	1354#	3324	3973							
CDFUF	360#	947	1129	1138	1161	1223	1273	1285	1327	1341	
	1382	2970	3015	3081	3129	3722	3783				
CDIFDF	3891	3957	3963#	3975							
CDIFI	1246	1247	1278	1284	1311	1312	1332#	1340			
CHECK	79#	591	870	1729	1954	4086	4182				
CHKMS	1454	3663	3739#	3750	3751	3758					
CHKMS2	3741	3745#									
CHKONL	3289#	3290	3293								
CHRCQ	3290#										
CHRPQ	3293#										
CIFINS	1314	1353#									
CINT	863										
CLINTR	3198#										
CLSA	791										
CLSK	783										
CMNSAF	1088	1098#									
CNINTR	3188#										
CORMAP	540	2136	2142	4412#	4423						
CUR	274#	304	330	364	398	417	435				
CURTSK	530#	938	1099	1157	1195	1199	1201	1202	1208	1222	
	1380	1401	1461	2683	2685	2912	2915	3123	3517	3532	

	3588	3774	3929	4002						
CURWT	1153#	1159	1189	2901	3575	3603				
C10	474#	3124								
C100	479#	1269	1323	2003	2023	2315	3493	4172		
C177	480#	2319	2726	2758	2796	4006	4034	4051	4079	4142
C2	467#	2000	2971	3537						
C200	481#	1120	1261	1357	2536	2800	3604			
C3	469#									
C37	475#	1533	1728	1918	2554	2734	2739	3338		
C377	482#	4112								
C4	471#	1912	2049	2055						
C40	476#									
C7	473#	1103	2605	2648	3498	3966	4177			
C70	477#	1236	1248	1947	2611	4107				
C7600	503#	2636	2748	2799	4018					
C77	478#	1107	1877	2011	2019	2244	2745	2963	3780	
C7700	501#	978	1122	1134	1240	1266	1305	2149	2520	2621
	3087	3777	3826							
C7760	499#									
C7770	497#									
C7772	495#									
C7773	493#	2054	4063							
C7774	491#									
C7775	489#									
C7776	487#	3823								
C7777	485#									
DCDIR	105#	106								
DCLR	2501	2590								
DCMA	2600#	2656								
DFPARM	2769	3311	3494#							
DF3	1794	1848	4268	4290#	4300					
DIMA	2599#	2647								
DIML	2598#	2613	2646							
DISINT	3181#									
DK8EP	37#	94	781	782	785	790				
DLAG	2545									
DLCA	2528									
DLDC	2506	2543								
DRST	2587									
DSKP	751									
DTA	731	732								
DTSF	733									
DXAL	2633	2655								
EAE	5#	7	1115	1128	1235	1303				
ERHLT	313#									
ERR	99	446	505	547						
ERROR	256	3381								
EXIT	3266#									
EXTALL	929#	1289	1346	3956						
EXTPRV	928#	3955								
FCCHAN	1734	4409#								
FCHECK	1230	1245	1310	2243#	2262	3127				
FCHEX	1380#	2268	2285							
FCRES	1636	1774	1780	1898	1942	2004	2027	2316#	4377	4387
FCSWAP	2264#	3105								
FDATA	1632	2027	2325	4060#						
FDCMAX	107#									
FDP	1643	2001	2027	2311	2313	2427#	4377	4387		

FLODTAB	1555	2432	4375	4376#	4384	4386					
FLDTEM	1981	1994	1997	2005	2031#						
FMININ	4064#										
FMONIN	1640#	4377									
FN	542	4275#	4277	4283	4284	4298					
FOCCUP	1645#	2027	4377	4387							
FPCR	845										
FPSF	843										
FPTP	841	842									
FPTR	850	851									
FRCR	854										
FRFLD	1872	1985	1991	2018#	2030	4072					
FRINTR	3206#										
FRSF	852										
FSTEXT	906	917#	2932								
FSTXT2	935	956#									
FTCHMS	2737	2765	2770	2778	2797	2809	2824#				
FWTMS	1450	1568#	4448	4480							
FWTRP	1449	1567#	4433	4465							
FWTSWP	1188	1570#									
FWTTM	1569#	3305									
FZREQ	1634#	1774	1780	1784	1818	1898	2027	4377	4387		
GET	395#	2724	2756	2806	3571	4004	4016				
GETFLD	1900	1979#	2012	4061							
GETFLF	1989	1994#									
GETFLP	1986#	1992									
GETFLX	2008	2010#									
GETFL2	1999	2009#									
GETMQ	9	11#	1146	1334							
GETTCB	3506	3712	3790	4078#	4085	4118	4147				
GN	4202#	4203	4208	4209	4210	4211	4215	4217	4231	4245	
	4255	4256	4278	4279	4282	4294	4295	4297	4303	4310	
	4317	4324									
GNA	4205	4217#									
GNALP	4219#	4264									
GNBR	4243#										
GNDEL	4242	4259#									
GNEX	4211#	4258									
GNEX2	4253#	4261									
GNFTEM	4206	4207	4213	4223	4224	4230	4244	4249	4260	4263	
	4276	4280	4281	4292	4293	4296	4330#				
GNNEXT	4235	4262#									
GNPR	4219	4220	4246	4247	4248	4250	4251	4253	4268	4289#	
GN2	2906	4302#	4307								
GN20	4090	4323#	4328								
GN3	1859	4309#	4314								
GN5	1190	3003	4316#	4321							
GPZCDF	2059	2060#	2091								
GPZCT	2065#	2067	2073								
GPZRIN	1915	2048#	2096	2159	2327						
GPZRI2	2062	2084#									
GPZTAD	2069	2070#	2071	2074	2089	2092					
HARDMQ	7	8#	9	11	451						
HCT	46#	58	59	60	61	62	63	64	65		
HEST											

IDLEIN	2685#	4498								
IDLP	2682	2686#	2698							
IDLP2	2696#	2700								
IDNS	2675#									
IDNSHL	2674#	2690								
IDPTR	2680#	2693	2696	2697	2699					
IDSW	2678#									
IDSWHL	2677#	2692								
EXIT	458#	682	691	700	709	718	727	792	803	846
	855	864	2585	2657	2935	3917				
INARG2	3014#	3020	3492	3923	4122					
INIT	263	264#								
INITIN	4513	4514#								
INRQEX	1538	1556#								
INRUNQ	1471	1531#	1557	1562	2816	2886	3828			
INS200	560	570#								
INTAC	253#	558	568	961	968	1084				
INTCT	578	580	582	609	877#					
INTDEF	29#									
INTFL	254#	565	573	958	965	1082				
INTPC	238#	950	962	971	977	983	984	986	987	1086
INTR	241	245	558	568#						
INTR2	575#	926								
INTSCD	891#	942	944							
INT0	519#	907	909	911	912	1193	1438	1469	1482	1483
	1501	1503	1544	1545	1547	1549	2280	3512	3518	3551
	3552	3558	3659	3667	3668	3670	3676	3743	3745	3753
INT1	520#	901	908	1439	1443	1451	1457	1459	1476	1489
	1496	1532	1534	1535	1542	1543	1548	1550	1551	1577
	1578	1583	1584	1585	1590	1595	1597	1598	2815	2885
	3509	3519	3793	3794	3801	3809	3811	3812	3813	3816
	3817	3824	3825							
INT2	521#	1444	1458	1467	1470	1473	1474	1477	1478	1481
	1491	1494	1502	1541	1546	1552	1555	1559	1581	1586
	1602	1604	3651	3746	3748	3755				
INT3	522#	1155	1582	1587	1588	1593	1596	1599	1603	3800
ITSAF	949	980	1081#							
JUMS	414#									
KB1	39#	53	686	687						
KB2	41#	62	704	705						
KB3	43#	71	722	723						
KCC1	54#									
KCC2	63#									
KCC3	72#									
KEEP	3298#									
KM8E	31#	859	860							
KRB1	56#									
KRB2	65#	708								
KRB3	74#	726								
KRS1	55#									
KRS2	64#									
KRS3	73#									
KSF1	53#									
KSF2	62#	706								
KSF3	71#	724								
LOCK	3254#									
MARG1	526#	1727	1779	1783	1819	1899	1921	2732	2733	2738
	3319	3501	3502	3511	3534	3549	3716	3895	3915	3960
	4012	4050	4052	4135						
MARG2	527#	3019	3505	3577	3919	3924	3943	4009	4010	4138

MAXDEV	44#	590	868								
MAXSTL	22#	4125	4366								
MCILF	3855	3859#									
MCILP	3846#	3858									
MCINTR	3844#	3861	3873	3888	3890	3893	3913				
MCROUT	3874	3896#	3920								
MONAC	252#	256	256	1085	1110	1252	1297	1316	1360	1464	
	2964	2976	3008	3064	3099	3677	3711	3779	3789	3853	
	4033	4049	4053	4062	4071	4146					
MONEX	928	1288	1345#	3525	3557	3679	4067	4187			
MONEX0	3556#	3561									
MONFL	255#	256	256	1083	1102	1121	1133	1239	1249	1250	
	1267	1268	1270	1293	1322	1324	1350	1355	3083	3497	
	3647	3648	3673	3674	3965	4176					
MONFUN	525#	3316	3327	3336	3495	3515	3523	3591	3606	3611	
	3685										
MONIN	1913#										
MONITO	454	3064#									
MONIT2	3089	3315#									
MONJMP	3339	3345#	3381								
MONOUT	1091#	3614									
MONPC	378#	1087	1112	1254	1298	1318	1362	3016	3018	3085	
	3086	3090	3101	3112	3315	3317	3318	3342	3570	4022	
	4023										
MONSAF	1099#	1164									
MONSCD	1093	1095	1149#								
MONSF2	1107#	1388									
MONSTA	3329	4425#									
MON0	513#	1105	1106	1276	1280	1330	1336	2870	2871	2888	
	2891	2896	2960	2962	2969	2974	2975	2978	2996	2998	
	3001	3067	3322	3533	3538	3589	3593	3649	3652	3775	
	3776	3778	3781	3782	3928	3930	4175	4181			
MON1	514#	2729	2730	2782	2783	2789	2790	2794	2798	2801	
	2879	2880	2884	2889	2890	2892	3655	3656	3658	3669	
	3719	3726	4179	4180	4186						
MON2	515#	3721	3730								
MON3	516#	1386	1785	1786	1825	1879	1880	1907	1946	1983	
	1987	1990	1996	2002	2009	2010	2021	2022	2028	2029	
	2050	2053	2056	2058	2246	2247	2276	2322	3104	4065	
	4066										
MON76	3328	3334#									
MQ	11	451#	1130	1335							
MSAPP	1453	1455	1481#								
MSAPP1	1491#	1497	1500								
MSAPP2	1493	1501#									
MSFREE	286#	2722									
MSIN	1456#										
MSNOD1	457	2994#									
MSNOD2	3002	3008#									
MSNREQ	2995	3003#									
MSQMAX	1487	1498	1508#								
MSREQ	285#										
MSTOAC	3667#										
MTFCT	2741	2802	2820#								
MTFFLD	2746	2751	2754	2767#	2773	2792					
MTFPAG	2743	2744	2747	2749	2759	2781	2787	2810	2825#		
MTFTCH	1197	2822	4355	4449#							
MTFTIN	2720#	2818	4453								
MTINIT	4364	4393	4394	4509#							
MTSDSK	2281	2576	2660	4361	4481#						

[illegible]

[illegible]

	4111	4126	4132								
SBYTER	247	1167#	1169								
SCA	1116										
SCDF	1214	1219#									
SCDF2	1272	1275	1326	1329#							
SCDF3	1274	1285	1328	1341#							
SCDINH	534#	933	1209	1221	1296	1352	1394	3078	3079		
SCDLP2	1211#	1402									
SCDREQ	532#	932	1206	1290	1347	1395	1561	3567			
SCED	940	1140	1148	1163	1205#	1397	2902	3579			
SCEDLP	1208#	1218									
SCEDNI	1384	1394#									
SCL	1119										
SDERCT	2470#	2565	2651								
SDEX	2394#	2422									
SDFCT	2405	2407	2448#								
SDGO	2391	2437	2479	2491	2602#	2609	2643				
SDIN	2397#	4485									
SDMSP	2390	2393	2396#	2399	2400						
SDNCT	2385	2387	2446#								
SDNORM	2383#	2403									
SDPTR	2469#	2480	2490	2512	2513	2516	2521	2522	2526	2527	
	2529	2568	2603	2604	2610	2614	2619	2620	2626	2627	
	2629										
SDSWP	2416	2421	2424#	2430	2440						
SDSW1	2417#										
SETAC	3677#	3687	3732	3818	3829	3897	4038	4055	4073	4143	
	4148										
SHOULD	256										
SIDLE	4348#	4495									
SIMINT	2484	3212#									
SINT	861										
SKPCHN	579	581	587	588	597	597	601	602	603#	603	
	608	608	678	687	696	705	714	723	732	741	
	750	766	782	798	814	828	842	851	860	868	
	3845										
SKPEND	590	591#	591	594							
SKPFIT	592	593#	593	594							
SKPSHF	602#	603									
SMSQ	3539	3594	3645#	3662	3672	3688	3689	3690	3692		
SMSQEX	3661	3684#									
SMSQLP	3655#	3664									
SMTFTC	4354#	4450									
SMTINI	4363#	4510									
SMTSDS	2771	4360#	4482								
SMTSWA	4351#	4435									
SMTTIM	4357#	4467									
SNDMS	3152#										
SNDREP	541	1433#	1479	1485	1506						
SNDWTR	2769	3165#									
STALL	3149#										
STATX	77#	81	577	593	597	609	876	891	941	1092	
	1149	1486	1495	1508	1608	2384	2404	2446	2448	2451	
	2470	2565	2649	2673	2686	2687	3326				

STTOMS	3887	3912	3922#	3926	3931						
SUF	1359										
SWAB	1256										
SWAPMS	2265	2269	2270	2272	2274	2275	2277	2279	2290#	2310	
	2314	2317	2318	2320	2321	2323	2328	2332	2401	2412	
	2415	2420	2688	4431							
SWBA	1264										
SWNODE	2312	2414	2418	2419	2425	2428	2431	2435	2436	2442#	
SWPOUT	3301#										
SYDISK	91#	753	758	769	774	2478	2594				
SYSHLT	551#	874	1731	1956	1993	2439	2780	3093	3384	3849	
	4088	4134	4184	4222							
SYTIME	94#	785	790	801	806	2926					
SYWAIT	2502	2507	2558	2570	2583#	2591					
TCF1	50#										
TCF2	59#	699									
TCF3	68#	717									
TCKLEN	84	85#	2674	2677	2694						
TIME	2865	2867	2923#								
TIMEIN	2864#	4470									
TIMEMS	793	2898	2922#	2936	4463						
TIMEOU	3149	3305	3306	3605#							
TLS1	52#										
TLS2	61#										
TLS3	70#										
TOQEX	2873	2898#									
TOQFN	2878	2888#									
TOQLP	2871#	2895	2897								
TOQLP2	2881	2896#									
TPC1	51#										
TPC2	60#										
TPC3	69#										
TQHEAD	2869	2907	2910	2918#							
TQIN	2905#	2916	3608								
TSBLOK	100#	2764									
TSF1	49#										
TSF2	58#	697									
TSF3	67#	715									
TSKSAF	1161#	1292	1349								
TSKSWT	954	977#									
TSTTM	3692#	3696	3697	3700	3708						
TSWAIT	3542	3603#									
TT1	38#	49	677	678							
TT2	40#	46	58	695	696						
TT3	42#	47	67	713	714						
T1ON	802	2930									
T1SKP	799										
UNLOCK	3257#										
VCALL	305	315	339	379#	951	3084					
VCDF	326#	2766									
VCDFTM	335	337	342#	3102							
VCDF1	3097#										
VCDIF	363#	365									
VCURIF	529#	1108	1242	1243	1307	1308	4024				
VERHLT	314#	3091									
VFBLOK	98#	99	99	100	2432						
VFLCT	2140	2152	2159#								
VFLESS	2138#	2154	2156	2157	2267						
VFLF	2151	2155#									
VFLOCK	1944#	1959	2007	2026							

[illegible]

_02173	2311	
_02174	2281	
_02175	2279	
_02370	2694	
_02371	2692	
_02372	2690	
_02373	2688	
_02374	2556	
_02375	2518	
_02376	2517	
_02377	2497	
_02573	2829	
_02575	2764	
_02577	2731	
_02765	3023	
_02766	3009	
_02770	2965	
_02775	2898	
_02777	2869	
_03166	3387	
_03167	3329	
_03173	3102	
_03174	3100	
_03175	3098	3111
_03176	3091	
_03177	3084	
_03364	3617	
_03366	3610	
_03573	3761	
_03574	3720	
_03576	3698	
_03762	3978	
_03764	3955	
_03766	3917	
_03767	3889	
_03770	3845	
_03773	3805	
_03777	3773	
_04165	4151	
_04166	4125	
_04177	4011	
_04375	4333	
_04376	4218	

V3A

```

1      /APP. A6, MC8 IN.
2      /INITIALISE AND START MC8 SYSTEM.
3
4      0001  FIELD 1
5      0200  *200
6      10200 4777' INIT,   JMS      PRTXT;TESTRT
7      10201 0600
8      10202 4776'       JMS      CRLF
9      10203 6202       CIF 0
10     10204 4775'       JMS      PG0IN  /GET GENERAL PAGE 0 IN.
11     10205 4774'       JMS      AVINIT
12     10206 1373       TAD      (INS200 /RESTORE START ADDRESS IN FLD0
13     10207 6201       CDF 0
14     10210 3772       DCA 1  (200
15     10211 6211       CDF 10
16     10212 4776' INIT2, JMS      CRLF
17     10213 6041       TSF
18     10214 5213       JMP      .-1
19
20     10215 6007       IFDEF PDP8E <
21                       CAF >
22     IFDEF PDP8I <
23                       P11IED
24                       P8ID
25                       6611
26                       6601
27                       DTCA DTXA
28                       P11SOF
29                       NOP
30                       P11CLF
31                       P11SIF
32                       NOP
33                       P11CLF
34                       P8CRD
35                       P8COF
36                       RRB
37                       PCF
38                       TCF
39                       CCF
40                       CRF
41                       PLCF
42                       FPCR
43                       FRCR
44                       PRCF
45                       BRCI
46                       KCC >
47
48     /INIT CLOCK
49     IFDEF DK8EP <
50     IFZERO SYTIME=DK8EP <
51     10216 7240       CLA CMA
52     10217 6130       CLZE
53     10220 7200       CLA
54     10221 1371       TAD      (-12
55     10222 6133       CLAB
56     10223 1370       TAD      (5210+12

```

```
56 10224 6132 CLOE
57 10225 7200 CLA
58 10226 6135 CLSA
59 10227 7200 CLA
60
61 >>
62 IFDEF MULTIP <
63 IFZERO SYTIME-MULTIP <
64 T1ON
65 >>
66 10230 6203 CIF CDF 0
67 10231 5767' JMP SCED
68
69 10367 0617
70 10370 5222
71 10371 7766
72 10372 0200
73 10373 6004
74 10374 0400
75 10375 5200
76 10376 1066
77 10377 1123
78 0400' PAGE
```

```

79      /AVAILLIST IN FLD 0.
80      /DURING ASSEMBLY AN AVAILLIST IS BUILT IN FLD 0.
81      /ITS FIRST NODE IS POINTED TO BY AVPT.
82      /EACH NODE HAS THE FOLLOWING LAYOUT:
83      /W0:   PTR TO NEXT NODE (0 INDICATES THE END)
84      /W1:   PTR TO LAST LOC OF NODE.
85      /EACH NODE IS AT LEAST 3 LOCS LONG.
86      /WE WILL INVERT THIS CHAIN AND COUNT ITS TOTAL LENGTH.
87      /NODES ALLOCATED IN THE RANGE 1-4000 ARE BROKEN SUCH
88      /THAT THEIR MAX LENGTH <=17 (NO TCBS <=4000).
89
90      10400  0000  AVINIT, 0      /INITIALIZE AVAILLIST.
91      10401  1377      TAD      (AVPT
92      10402  3307      DCA      AVPTR /PTR TO NEXT NODE.
93      10403  1376      TAD      (20 /TCB OF MTINIT WILL BE ADDED TO
94      10404  3312      DCA      AVCT /AVAILLIST. INITIALIZE COUNTER
95      10405  3311      DCA      AVOLD /WE WILL INVERT THE CHAIN AS IT IS
96      /IN CORE, SO THIS NODE WILL BECOME
97      /THE LAST ONE.
98      10406  6201      CDF 0
99      10407  1307  AVINLP, TAD      AVPTR /PTR TO NEW NODE.
100     10410  7450      SNA
101     10411  5233      JMP      AVIN2 /NO MORE NODES TO DO.
102     10412  7001      IAC
103     10413  3310      DCA      AVPTR2 /MAKE PTR TO PTR TO NODE'S LAST LOC,
104     10414  1307      TAD      AVPTR /FIRST LOC OF NODE
105     10415  7041      CIA
106     10416  7001      IAC      /-(FIRST LOC) +1
107     10417  1710      TAD      AVPTR2 /-(FIRST LOC) +1 +(LAST LOC)=LENGTH.
108     10420  1312      TAD      AVCT
109     10421  3312      DCA      AVCT /ADDED LENGTH TO COUNTER.
110     10422  1707      TAD      AVPTR
111     10423  3310      DCA      AVPTR2 /PTR TO NEXT NODE.
112     10424  1311      TAD      AVOLD /INVERT CHAIN, STORE PTR TO PRECESSOR,
113     10425  3707      DCA      AVPTR
114     10426  1307      TAD      AVPTR /SAFE THE NEW PRECESSOR
115     10427  3311      DCA      AVOLD
116     10430  1310      TAD      AVPTR2 /NEXT NODE BECOMES NEW NODE.
117     10431  3307      DCA      AVPTR
118     10432  5207      JMP      AVINLP
119     10433  1311  AVIN2, TAD      AVOLD
120     10434  3775      DCA      (AVHEAD /PASS HEAD OF AVAILLIST TO MC8,
121     10435  1311      TAD      AVOLD
122     10436  3307      DCA      AVPTR /KEEP LOW NODES SHORTER THAN 20,
123     10437  1307  AVNLP, TAD      AVPTR /MORE TO DO?
124     10440  7450      SNA
125     10441  5300      JMP      AVEND
126     10442  7710      SPA CLA /LOW NODE?
127     10443  5275      JMP      AVNLP2 /NO
128     10444  1307      TAD      AVPTR
129     10445  7001      IAC      /PTR TO PTR TO LAST LOC
130     10446  3310      DCA      AVPTR2
131     10447  1307      TAD      AVPTR
132     10450  7041      CIA
133     10451  7001      IAC      /-(FIRST LOC) +1

```

134	10452	1710	TAD	AVPTR2	/LENGTH
135	10453	7100	CLL		
136	10454	1374	TAD	(-20	/LOW NODE AND TOO LONG?
137	10455	7620	SNL	CLA	
138	10456	5275	JMP	AVNLP2	/NOT TOO LONG,
139	10457	1307	TAD	AVPTR	/TOO LONG NODE, BREAK OFF 16 WORDS.
140	10460	1373	TAD	(16	
141	10461	3311	DCA	AVOLD	/JUST A PTR TO NEW NODE.
142	10462	1707	TAD	AVPTR	
143	10463	3711	DCA	AVOLD	/NEW NODE IN CHAIN,
144	10464	2311	ISZ	AVOLD	
145	10465	1710	TAD	AVPTR2	/GET PTR TO LAST LOC
146	10466	3711	DCA	AVOLD	/AND STORE
147	10467	7040	CMA		
148	10470	1311	TAD	AVOLD	
149	10471	3707	DCA	AVPTR	/UPDATE PRECESSOR.
150	10472	7344	ACM2		
151	10473	1311	TAD	AVOLD	
152	10474	3710	DCA	AVPTR2	
153	10475	1707	AVNLP2, TAD	AVPTR	/NEXT NODE.
154	10476	3307	DCA	AVPTR	
155	10477	5237	JMP	AVNLP	
156	10500	6211	AVEND, CDF	10	
157	10501	4772	JMS	CRLF	
158	10502	4771	JMS	PRTXT;TEAV	
159	10503	0623			
160	10504	1312	TAD	AVCT	
161	10505	4770	JMS	PRDEC	
162	10506	5600	JMP	AVINIT	
163					
164	10507	0000	AVPTR,	0	
165	10510	0000	AVPTR2,	0	
166	10511	0000	AVOLD,	0	
167	10512	0000	AVCT,	0	
168					
169	10570	1000			
170	10571	1123			
171	10572	1066			
172	10573	0016			
173	10574	7760			
174	10575	4303			
175	10576	0020			
176	10577	5142			
177		0600	PAGE		

178	10600	4040	TESTRT, TEXT	'	*** MULTI CORE 8 ***'
179	10601	4040			
180	10602	4040			
181	10603	4040			
182	10604	4040			
183	10605	4040			
184	10606	4040			
185	10607	4040			
186	10610	5252			
187	10611	5240			
188	10612	1525			
189	10613	1424			
190	10614	1140			
191	10615	0317			
192	10616	2205			
193	10617	4070			
194	10620	4052			
195	10621	5252			
196	10622	0000			
197	10623	1116	TEAV, TEXT	'INITIAL SPACE IN AVAILLIST: '	
198	10624	1124			
199	10625	1101			
200	10626	1440			
201	10627	2320			
202	10630	0103			
203	10631	0540			
204	10632	1116			
205	10633	4001			
206	10634	2601			
207	10635	1114			
208	10636	1411			
209	10637	2324			
210	10640	7240			
211	10641	0000			
212					
213		1000	PAGE		

```
214 /ROUTINE IN FLD0 TO GET PAGE 0 INTO FLD 1,
215 /NOTE: THE LAST NODE IN THE AVAIL LIST MUST
216 /BE LONG ENOUGH TO HOLD THIS ROUTINE,
217 0000 FIELD 0
218 5144 *AVPT*2
219 5200 PAGE
220 05200 0000 PG0IN, 0
221 05201 3217 DCA PT0
222 05202 3220 DCA PT1
223 05203 1222 TAD M150
224 05204 3221 DCA CTX
225 05205 6201 PG0LP, CDF 0
226 05206 1617 TAD 1 PT0
227 05207 6211 CDF 10
228 05210 3620 DCA 1 PT1
229 05211 2217 ISZ PT0
230 05212 2220 ISZ PT1
231 05213 2221 ISZ CTX
232 05214 5205 JMP PG0LP
233 05215 6213 CIF CDF 10
234 05216 5600 JMP 1 PG0IN
235 /WARNING:!!!!!!
236 /THE "CIF CDF CUR"S ARE NOT CORRECT.
237
238 05217 0000 PT0, 0
239 05220 0000 PT1, 0
240 05221 0000 CTX, 0
241 05222 7630 M150, -150
```

```

242      /MTINIT.
243      /MONITOR TASK FOR INITIALISATION.
244      /BE AWARE OF ERRORS IN PAGE 0.
245      /THE "CIF CDF CUR"S ARE INCORRECT!!!
246
247      0001  FIELD 1
248      2000  *INITIN
249      12000 4054  CALL;  REQSTL  /INSERT TASKLOADER IN STL,
250      12001 0041
251      12002 0001      1;1437      /ALLOACTED IN TASKLIBRARY FROM BLOCK 1,
252      12003 1437
253      12004 3253      DCA      TSLDX
254      12005 4014      MSREQ
255      12006 3236      DCA      MSP1
256      12007 1236      TAD      MSP1
257      12010 3110      DCA      X
258      12011 6201      CDF 0
259      12012 1377      TAD      (0714  /GL
260      12013 3510      DCA I    X
261      12014 2110      ISZ      X
262      12015 1376      TAD      (1702  /OB
263      12016 3510      DCA I    X
264      12017 2110      ISZ      X
265      12020 1375      TAD      (0114  /AL
266      12021 3510      DCA I    X
267      12022 6211      CDF 10    /WE KNOW IN WHICH FLD WE ARE,
268      12023 1236      TAD      MSP1
269      12024 4246      JMS      SEND  /SEND MS TO LOAD FAMILY OF GLOBAL TASKS,
270      12025 4014      MSREQ
271      12026 3245      DCA      MSX
272      12027 6201      CDF 0
273      12030 1253      TAD      TSLDX  /PASS STATIC TASKNUM OF TSLD
274      12031 3645      DCA I    MSX
275      12032 1245      TAD      MSX
276      12033 4246      JMS      SEND  /PASS STSK# OF TSLD TO TASKLOADER,
277      12034 4054      CALL;    WTRP
278      12035 0005
279      12036 0000  MSP1,      0      /WAIT UNTIL LOADING COMPLETED,
280      12037 4014      MSFREE     /GET COMPLETION STATUS OF TSLD,
281      12040 7440      SZA
282      12041 5244      JMP      LDER  /LOAD ERROR,
283      12042 4054      CALL
284      12043 0054      EXIT
285      12044 4023  LDER,      ERHLT
286
287      12045 0000  MSX,      0
288
289      12046 0000  SEND,      0      /SEND MS TO TASKLOADER,
290      12047 3252      DCA      MSP
291      12050 4054      CALL
292      12051 0003      SNDMS
293      12052 0000  MSP,      0
294      12053 0000  TSLDX,      0
295      12054 5646      JMP      SEND

```

\$

PAL8-V9H 02/14/77 PAGE 9

296	12175	0114
297	12176	1702
298	12177	0714
299		

\$

\$

PAL8-V9H 02/14/77 PAGE 10

AVCT	0512
AVEND	0500
AVINIT	0400
AVINLP	0407
AVIN2	0433
AVNLP	0437
AVNLP2	0475
AVOLD	0511
AVPTR	0507
AVPTR2	0510
CRLF	1066
CTCTST	1111
CTX	5221
DECBIN	1043
INIT	0200
INIT2	0212
LDER	2044
MSP	2052
MSP1	2036
MSX	2045
M150	5222
PG0IN	5200
PG0LP	5205
PRDBOX	1041
PRDEC	1000
PRDECI	1015
PRDEC0	1007
PRDEC1	1011
PRDEC2	1013
PRHALF	1140
PRKCT	1104
PRKLP	1052
PRKTM	1103
PROCT	1046
PRTXLP	1127
PRTXT	1123
PT0	5217
PT1	5220
SEND	2046
SPACE	1105
TEAV	0623
TESTRT	0600
TSLDX	2053
TYPE	1074
ZFLAG	1042

ERRORS DETECTED: 0
LINKS GENERATED: 9
300

P8CRD	33					
P8ID	23					
REQSTL	250					
SCED	67					
SEND	269	276	289#	295		
SNDMS	292					
SYTIME	49	62				
TEAV	159	197#				
TESTRT	7	178#				
TSLDX	253	273	294#			
TION	63					
WTRP	278					
X	257	260	261	263	264	266
_10370	55					
_10371	53					
_10372	14					
_10373	12					
_10573	140					
_10574	136					
_10575	120					
_10576	93					
_10577	91					
_12175	265					
_12176	262					
_12177	259					

V3A

```

1      /APP. B1. RTTY,TK
2      /READ CHARS FROM CONSOLE TELETYPE, BUILD A LINE OF TEXT
3      /AND SEND IT TO WTTY.
4      /TO BE ASSEMBLED WITH MC8PAL.
5      /
6      /CHARS ARE READ FROM THE CONSOLE TELETYPE, A LINE OF TEXT IS
7      /ASSEMBLED, RUBOUTS ERASE THE LAST CHAR TYPED,
8      /MULTIPLE RUBOUTS ARE ALLOWED,
9      /THE LINE IS TERMINATED BY A CARRIAGE RETURN (215).
10     /THE TEXT LINES ARE STORED IN A BUFFER REQUESTED FROM
11     /THE PAGE ALLOCATOR.
12     /CHARACTERS ARE ECHOED BY SENDING CHARACTER-MESSAGES TO WTTY,
13     /WHEN A LINE IS COMPLETED (215 SEEN), THE LINE IS SENT TO WTTY
14     /USING ALINE-MESSAGE,
15     /WTTY RETURNS THE BUFFER TO THE PAGE ALLOCATOR.
16     /
17     /LAYOUT OF CHARACTER-MESSAGE:
18     /      W1 B4-11      CHARACTER
19     /      W2            -1
20     /LINE-MESSAGE:
21     /      W1 B0-4      PAGE ADDRESS OF TEXT BUFFER
22     /      W1 B6-11     VIRTUAL FIELDNUMBER OF TEXT BUFFER
23     /      W2            0
24
25     0000  *0
26     00000 4002 4002 /CONTENTS OF ENTRY IN STL: PRIORITY 4, LENGTH 2,
27     00001 4000 4000 /AUTOSTART. IS STARTED WHEN LOADED.
28     00002 2405 TASKNAME TEST,RTTY
29     00003 2324
30     00004 0000
31     00005 2224
32     00006 2431
33     0203 PAGE
34
35     0107 CHAR=BASE /STORE CHAR IN BASE REGISTER.
36
37     00203 0000 0 /STOP RELOCATION.
38     /INITIALISATION,
39     00204 3211 START, DCA MSCL /ARRIVES WITH MSPTR IN AC,
40     00205 1377 TAD (KSF /CLAIM KEYBOARD INTSLOT
41     00206 4054 CALL; CLINTR
42     00207 0023
43     00210 6036 MSCL, KRB /CLEAR AND READ
44     00211 0000 0
45     00212 1211 TAD MSCL
46     00213 3600 DCA 1 )MSRD /HAND MESSAGE TO READ ROUTINE.
47
48     /START AT A NEW LINE.
49     00214 4054 NEWLIN, CALL; REQ PAG /REQUEST ONE PAGE BUFFER FROM PAGE
50     00215 0031
51     /ALLOCATOR.
52     00216 0001 1
53     00217 4054 CALL; REQ CDF /PREPARE FOR FIELD ACCESS USING THE VRCDF
54     00220 0052
55     /PSEUDO INSTRUCTION

```

```

56 00221 3314      DCA      BUFAD      /STORE BUFFER ADDRESS
57 00222 1314      TAD      BUFAD      /SET BUFFER POINTER IN X, SET COUNTER,
58 00223 0146      AND      C7600
59 00224 3110      DCA      X
60 00225 1376      TAD      (-177      /ALLOW 127 CHARS ON A LINE,
61 00226 3315      DCA      CT
62
63                /READ CHARACTER LOOP.
64 00227 4601      CHARLP, JMS      )CHREAD /READS A CHAR, STORES IN CHAR,
65 00230 1107      TAD      CHAR      /IS IT RUBOUT
66 00231 1375      TAD      (-377
67 00232 7450      SNA
68 00233 5277      JMP      RUB      /YES IT IS.
69 00234 1374      TAD      (377-215 /IS IT CR?
70 00235 7650      SNA CLA
71 00236 5250      JMP      LINEND /YES, TERMINATE LINE.
72                /NO SPECIAL CHARS, ECHO AND INSERT IN BUFFER.
73 00237 1107      TAD      CHAR
74 00240 4602      JMS      )ECHO
75 00241 1107      TAD      CHAR
76 00242 4020      VRCDF      /CDF TO BUFFER FIELD.
77 00243 3510      DCA      X
78 00244 4051      CDFCUR
79 00245 2110      ISZ      X
80 00246 2315      ISZ      CT      /BUFFER FULL?
81 00247 5227      JMP      CHARLP /NO.
82                /FULL BUFFER; FALL INTO LINE END.
83
84 00250 1373      LINEND, TAD      (215 /ECHO CR.
85 00251 4602      JMS      )ECHO
86 00252 1373      TAD      (215 /INSERT CR INTO BUFFER
87 00253 4020      VRCDF
88 00254 3510      DCA      X
89 00255 4051      CDFCUR
90                /SEND THE LINE-MESSAGE.
91 00256 4014      MSREQ
92 00257 3110      DCA      X
93 00260 1110      TAD      X
94 00261 3272      DCA      MSSD      /PASS TO SEND ROUTINE.
95 00262 1314      TAD      BUFAD
96 00263 6201      CDF 0
97 00264 3510      DCA      X
98 00265 2110      ISZ      X
99 00266 3510      DCA      X
100 00267 4051      CDFCUR
101 00270 4054      CALL;      SNDWTR
102 00271 0007
103 00272 0000      MSSD,      0
104 00273 0000      LCTASK WTTY
105                /NOTE, WTTY RETURNS THE BUFFER.
106 00274 4014      MSFREE      /RETURN THE MESSAGE.
107 00275 7200      CLA      /REMOVE RUBBISH
108 00276 5214      JMP      NEWLIN
109
110                /RUBOUT. ECHO "\" IF A CHAR IS ACTUALLY DELETED.

```

111	00277	1110	RUB,	TAD	X	
112	00300	0132		AND	C177	/FETCH NUMBER OF CHARS IN BUFFER,
113	00301	7650		SNA	CLA	
114	00302	5227		JMP	CHARLP	/EMPTY BUFFER,
115	00303	1372		TAD	("\"	
116	00304	4602		JMS	I	)ECHO
117	00305	7240		CLA	CMA	
118	00306	1110		TAD	X	
119	00307	3110		DCA	X	/DECREMENT POINTER
120	00310	7240		CLA	CMA	
121	00311	1315		TAD	CT	
122	00312	3315		DCA	CT	/DECREMENT COUNTER
123	00313	5227		JMP	CHARLP	
124						
125	00314	0000	BUFAD,	0		
126	00315	0000	CT,	0		
127						
128	00200	0403				
129	00201	0400				
130	00202	0412				
131	00372	0334				
132	00373	0215				
133	00374	0162				
134	00375	7401				
135	00376	7601				
136	00377	6031				
137		0400	PAGE			

```

138      /READ CHARACTER.
139      00400 0000 CHREAD, 0 /THIS 0 STOPS RELOCATION.
140      00401 4054 CALL; WTRP /WAIT FOR REPORT OF INTERRUPT MESSAGE
141      00402 0005
142      00403 0000 MSRD, 0
143      00404 7200 CLA /ERASE MSPTR FROM AC.
144      00405 6201 CDF 0 /FETCH CHAR FROM MESSAGE
145      00406 1603 TAD I MSRD
146      00407 4051 CDFCUR
147      00410 3107 DCA CHAR
148      00411 5600 JMP I CHREAD
149
150      /ECHO THE CHARACTER IN AC.
151      00412 0000 ECHO, 0
152      00413 3234 DCA TEM
153      00414 4014 MSREQ
154      00415 3231 DCA MSSD2
155      00416 1234 TAD TEM /FETCH CHAR AGAIN
156      00417 6201 CDF 0
157      00420 3631 DCA I MSSD2 /STORE MESSAGE W1
158      00421 7001 IAC
159      00422 1231 TAD MSSD2
160      00423 3234 DCA TEM /POINTER MESSAGE W2
161      00424 7240 CLA CMA
162      00425 3634 DCA I TEM /-1 TO MESSAGE W2.
163      00426 4051 CDFCUR
164      00427 4054 CALL; SNDMS+NONREP /DONT WAIT FOR THIS REPORT.
165      00430 2003
166      00431 0000 MSSD2, 0
167      00432 0000 LCTASK WTTY
168      00433 5612 JMP I ECHO
169
170      00434 0000 TEM, 0

```

\$

MC8PAL-V1A 02/14/77 PAGE 5

171 \$

\$

MC8PAL-V1A 02/14/77 PAGE 6

BUFAD	0314
CHAR	0107
CHARLP	0227
CHREAD	0400
CT	0315
ECHO	0412
LINEND	0250
MSCL	0211
MSRD	0403
MSSD	0272
MSSD2	0431
NEWLIN	0214
RUB	0277
START	0204
TEM	0434

ERRORS DETECTED: 0
LINKS GENERATED: 0
172

BASE	35																			
BUFAD	56	57	95	125#																
CALL	41	49	53	101	140	164														
CDFCUR	78	89	100	146	163															
CHAR	35#	65	73	75	147															
CHARLP	64#	81	114	123																
CHREAD	64	139#	148																	
CLINTR	42																			
CT	61	80	121	122	126#															
C177	112																			
C7600	58																			
ECHO	74	85	116	151#	168															
LCTASK	104	167																		
LINEND	71	84#																		
MSCL	39	44#	45																	
MSFREE	106																			
MSRD	46	142#	145																	
MSREQ	91	153																		
MSSD	94	103#																		
MSSD2	154	157	159	166#																
NEWLIN	49#	108																		
NONREP	165																			
REQCDF	54																			
REQPAG	50																			
RTTY	28																			
RUB	68	111#																		
SNDMS	165																			
SNDWTR	102																			
START	39#																			
TASKNA	28																			
TEM	152	155	160	162	170	171#														
TEST	28																			
VRCDF	76	87																		
WTRP	141																			
WTTY	104	167																		
X	59	77	79	88	92	93	97	98	99	111										
	118	119																		
_00200	46																			
_00201	64																			
_00202	74	85	116																	
_00372	115																			
_00373	84	86																		
_00374	69																			
_00375	66																			
_00376	60																			
_00377	40																			

V3A

```

1      /APP. B2. WTTY.TK
2      /WRITE CHARACTERS ON CONSOLE TELETYPE
3      /TO BE ASSEMBLED WITH MC8PAL.
4      /THIS TASK OPERATES IN TWO MODES:
5      /LINEMODE, ECHOING A FULL LINE OF TEXT, TERMINATED BY
6      /      A CARRIAGE RETURN (215);
7      /CHARACTERMODE, ECHOING A SINGLE CHARACTER,
8      /MODE CHANGES CAN OCCUR AFTER EACH CARRIAGE RETURN.
9      /IN CHARACTERMODE THE TASK REMAINS CLAIMED UNTIL THE
10     /NEXT CARRIAGE RETURN,
11     /AFTER EACH CARRIAGE RETURN A LINEFEED IS ADDED,
12     /IN LINEMODE THE TEXTBUFFER IS RETURNED TO THE PAGE ALLOCATOR,
13
14     /MESSAGE LAYOUT:
15     /CHARACTERMODE: W1 B4-11 CHARACTER
16     /                  W2      -1
17     /LINEMODE:      W1 B0-4  PAGE ADDRESS OF TEXT LINE
18     /                  W1 B6-11 VFLD# OF TEXT LINE
19     /                  W2      0
20
21     0000      *0
22     00000 4001 4001      /CONTENTS OF ENTRY IN STL: PRIORITY 4, LENGTH 1.
23     00001 0000 0000      /NO AUTOSTART
24     00002 2405 TASKNAME TEST.WTTY
25     00003 2324
26     00004 0000
27     00005 2724
28     00006 2431
29     0200 PAGE
30
31     00200 0000      0      /STOP RELOCATION
32     00201 3316 START, DCA      MSREP /SAVE MESSAGE PTR
33
34     /INITIALIZATION: CLAIM INTSLOT OF TELETYPE.
35
36     00202 4014      MSREQ
37     00203 3210      DCA      MSCL
38     00204 1377      TAD      (TSF
39     00205 4054      CALL
40     00206 0023      CLINTR
41     00207 6042      TCF
42     00210 0000 MSCL,      0      /PTR TO INTERRUPT MESSAGE
43     00211 1210      TAD      MSCL
44     00212 3270      DCA      MSWR      /PASS MESSAGE PTR TO WRITE ROUTINE
45     00213 1316      TAD      MSREP /FETCH MESSAGEPTR IN AC
46     00214 5217      JMP      NEWMOD+2 /FALL INTO NEWMODE LOOP
47
48     /WE HAVE SEEN A CARRIAGE RETURN, READY TO CHANGE MODE,
49     NEWMOD, CALL
50     00216 0014      WTMS      /TERMINATE PREVIOUS CLAIM
51     00217 4273      JMS      MSREAD /READ MESSAGE INTO W1 AND W2
52     00220 2314      SZ      W2
53     00221 5234      JMP      LINMOD /LINEMODE
54
55     /CHARACTERMODE

```

```

56
57 00222 1313 CHRMOD, TAD W1
58 00223 4264 JMS WRITE
59 00224 1313 TAD W1
60 00225 1376 TAD (-215
61 00226 7650 SNA CLA
62 00227 5261 JMP MODEND /LAST CHAR, ECHO LF.
63 00230 4054 CALL
64 00231 1014 WTMS+KEEP /KEEP CLAIMED.
65 00232 4273 JMS MSREAD
66 00233 5222 JMP CHRMOD
67
68 /LINEMODE
69
70 00234 1313 LINMOD, TAD W1
71 00235 4054 CALL
72 00236 0052 REQCDF /PREPARE FOR ACCESSING THE TEXTLINE
73 /WITH THE VRCDF PSEUDO INSTRUCTION.
74 00237 0146 AND C7600
75 00240 3110 DCA X /SET BUFFER PTR
76 00241 4020 LNLP, VRCDF
77 00242 1510 TAD I X
78 00243 4051 CDFCUR
79 00244 2110 ISZ X
80 00245 3312 DCA CHAR
81 00246 1312 TAD CHAR
82 00247 4264 JMS WRITE
83 00250 1312 TAD CHAR
84 00251 1376 TAD (-215
85 00252 7640 SZA CLA
86 00253 5241 JMP LNLP
87 00254 1313 TAD W1 /CONTAINS STILL PAGE ADDRESS + VFLD#
88 00255 4054 CALL
89 00256 0033 RTNPAG /RETURN TEXT BUFFER
90 00257 0001 1
91 00260 7200 CLA
92
93 /READY WITH BOTH MODES, ECHO LF
94 00261 1375 MODEND, TAD (-212
95 00262 4264 JMS WRITE
96 00263 5215 JMP NEWMOD /READY. START NEW MODE
97
98 00264 0000 WRITE, 0
99 00265 6044 TPC
100 00266 4054 CALL
101 00267 0005 WTRP
102 00270 0000 MSWR, 0
103 00271 7200 CLA
104 00272 5664 JMP I WRITE
105
106
107 00273 0000 MSREAD, 0
108 00274 3310 DCA MSP
109 00275 1310 TAD MSP
110 00276 3315 DCA TEMP

```

111	00277	6201		CDF 0	
112	00300	1715		TAD I	TEMP
113	00301	2315		ISZ	TEMP
114	00302	3313		DCA	W1
115	00303	1715		TAD I	TEMP
116	00304	4051		CDFCUR	
117	00305	3314		DCA	W2
118	00306	4054		CALL	
119	00307	0013			RP
120	00310	0000	MSP,		0
121	00311	5673		JMP I	MSREAD
122					
123	00312	0000	CHAR,	0	
124	00313	0000	W1,	0	
125	00314	0000	W2,	0	
126	00315	0000	TEMP,	0	
127	00316	0000	MSREP,	0	

\$

MC8PAL-V1A 02/14/77 PAGE 4

128	00375	0212
129	00376	7563
130	00377	6041
131	\$	

\$

MC8PAL-V1A 02/14/77 PAGE 5

CHAR	0312
CHRMOD	0222
LINMOD	0234
LNLP	0241
MODEND	0261
MSCL	0210
MSP	0310
MSREAD	0273
MSREP	0316
MSWR	0270
NEWMOD	0215
START	0201
TEMP	0315
WRITE	0264
W1	0313
W2	0314

ERRORS DETECTED: 0
LINKS GENERATED: 0
132

CALL	39	49	63	71	88	100	118
CDFCUR	78	116					
CHAR	80	81	83	123#			
CHRMOD	57#	66					
CLINTR	40						
C7600	74						
KEEP	64						
LINMOD	53	70#					
LNLP	76#	86					
MODEND	62	94#					
MSCL	37	42#	43				
MSP	108	109	120#				
MSREAD	51	65	107#	121			
MSREP	32	45	127#				
MSREQ	36						
MSWR	44	102#					
NEWMOD	46	49#	96				
REQCDF	72						
RP	119						
RTNPAG	89						
START	32#						
TASKNA	24						
TEMP	110	112	113	115	126#		
TEST	24						
VRCDF	76						
WRITE	58	82	95	98#	104		
WTMS	50	64					
WTRP	101						
WTTY	24						
W1	57	59	70	87	114	124#	
W2	52	117	125#				
X	75	77	79				
_00375	94						
_00376	60	84					
_00377	38						

V3A

```

      /APP. A0.  CONFIGURATION FILE.
      /PROCESSOR TYPE.
0001  PDP8E=1
      /PDP8I=1
0001  EAE=1

0001  IFDEF EAE <HARDMG=1>
0001  IFDEF PDP8E <HARDMG=1>
7701  IFDEF  HARDMG  <GETMQ=CLA MQA; STORMQ=ML>
7421
      IFNDEF HARDMG  <GETMQ=TAD MQ; STORMQ=DCA MQ>

      /INDICATE ACTUAL CORESIZE (HIGHEST ACTUAL FIELD)
0007  ACTMAX=7

      /INDICATE VIRTUAL CORESIZE (HIGHEST VIRTUAL FIELD)
0077  VIRMAX=77          /MUST BE <=77

      /INDICATE LENGTH OF STATIC TASK LIST (HIGHEST ST#)
      /THIS VALUE DETERMINES THE MAX NUMBER OF TASKS THAT
      /CAN RUN SIMULTANEOUSLY.
0077  MAXSTL=77          /MUST BE <=177

      /CONFIGURATE SKIPCHAIN.
      /INDICATE YOUR DEVICES IN THE ORDER OF THEIR
      /INTERRUPT PRIORITY.
      NOPUNCH          /IDEA OF MULTI 8.
0000  *0
      INTDEF,
00000 0000  RK8E,  0
00001 0000  KM8E,  0
      /MULTIP,          0          /MULTIPLEXER CLOCK.
00002 0000  PDP8IE, 0;0      /TWO DEVICES !!!!!!!!!!!
00003 0000
      /PDP11, 0;0
      /RF08,  0
00004 0000  DK8EP,  0
00005 0000  TT1,    0
00006 0000  KB1,    0
00007 0000  TT2,    0
00010 0000  KB2,    0
00011 0000  TT3,    0
00012 0000  KB3,    0
      MAXDEV,
      ENPUNCH
0370  IFDEF TT2 <HCT=6420-6030>
0430  IFDEF TT3 <HP=6460-6030>

6041  IFDEF TT1 <TSF1=TSF;TCF1=TCF;TPC1=TPC;TLS1=TLS>
6042
6044
6046
6031  IFDEF KB1 <KSF1=KSF;KCC1=KCC;KRS1=KRS;KRB1=KRB>
6032
6034

```

6036

6431 IFDEF TT2 <TSF2=TSF+HCT;TCF2=TCF+HCT;TPC2=TPC+HCT;TLS2=TLS+HCT>

6432

6434

6436

6421 IFDEF KB2 <KSF2=KSF+HCT;KCC2=KCC+HCT;KRS2=KRS+HCT;KRB2=KRB+HCT>

6422

6424

6426

6471 IFDEF TT3 <TSF3=TSF+HP;TCF3=TCF+HP;TPC3=TPC+HP;TLS3=TLS+HP>

6472

6474

6476

6461 IFDEF KB3 <KSF3=KSF+HP;KCC3=KCC+HP;KRS3=KRS+HP;KRB3=KRB+HP>

6462

6464

6466

/SYSTEM STATISTICS WANTED?

0001 STATX=1

/EXTRA ERROR CHECK WANTED?

0001 CHECK=1

IFDEF STATX <

/SET VALUE FOR MEASURING TIME IN IDLELOOP.

DECIMAL

6331 IFDEF PDP8E < TCKLEN=3289 /ABOUT 100.000/30.4 FOR PDP8/E>

IFDEF PDP8I < TCKLEN=2777 /ABOUT 100.000/36 FOR PDP8/I>

/USED FOR IDLETIME STATISTICS.

OCTAL

>

/SELECT SYSTEM DISK.

0000 SYDISK=RK8E

/SYDISK=RF08

/SELECT SYSTEM CLOCK.

0004 SYTIME=DK8EP

/SYTIME=MULTIP

/CONFIGURATE LOGICAL UNIT 0 OF SYSTEM DISK.

4000 VFBLOK=4000 /START BLOCK OF VFLDSLOTS.

IFNZRO VFBLOK-17 <ASS ERR! VFBLOK MUST BE MULTIPLE OF 20>

6000 TSBLOK=VIRMAX+1+20+VFBLOK /STARTING BLOK OF TASK LIBRARY.

/SOME STUFF FOR PROGRAMS USING THE TASKLIBRARY.

/NOT REQUIRED FOR MC8!!!!!!!!!!!!

/ASSUME DIRECTORY SWAPPED IN; TSBLOK+1 STARTS AT LOC200.

0200 DCDIR=200

0400 NRT=DCDIR+200

5000 FDCMAX=5000

/FIRST FREE LOC AFTER DIRECTORIES.

/APP. C. MC8 TASKLOADER TASK,

```

0000 0000 *0
0000 1437 1437 /PRIO=1, ZREQ, LENGTH=37.
0000 0000 0000 /NO AUTOSTART,
0000 0714 TASKNAME GLOBAL.TSLD
0000 1702
0000 0114
0000 2423
0000 1404

```

0200 PAGE

/R VAN VLIET, MATH' CENTR', A'DAM; 6/1/76.

/TASK TO LOAD AND UNLOAD TASKS,

/RUNNING A TASK IN THE MC8 SYSTEM GOES IN THE FOLLOWING STEPS:

- /A, A TWO-WORD ENTRY IN THE STATIC TASK LIST IS REQUESTED AND
- / ASSIGNED TO THE TASK, THE SOCALLED STATIC TASKNUMBER
- / -STSK#- WHICH IDENTIFIES THAT ENTRY IN THE STATIC TASK
- / LIST, IF P IS WO OF THE ENTRY, STSK#=(P/2)^177.
- / IN THE ENTRY SUFFICIENT INFO IS STORED TO START UP THE
- / TASK (ITS BLOCKNUMBER ON DISK, LENGTH, PRIORITY ETC.).
- /B, THE STATIC TASK NUMBER CAN BE USED TO ADDRESS A MS TO THE
- / TASK, AS SOON AS SUCH A MS IS SENT, THE SYSTEM ALLOCATES
- / A TASK CONTROLBLOCK FOR THE TASK AND ADDS IT TO THE RUNQ,
- /C, WHEN THE RUNQ IS SCANNED THE TASK IS FOUND TO BE RUNNABLE
- / BUT ITS CODE IS STILL ON DISK. THE SYSTEM REMOVES IT FROM
- / THE RUNQ AND SENDS A MS TO THE MONITOR FETCH TASK MTFTCH,
- /D, MTFTCH ALLOCATES SPACE FOR THE TASKS CODE, SWAPS IT IN
- / CORE, DOES THE RELOCATION AND REINSERTS IT IN THE RUNQ,
- /E, FINALLY THE RUNQ IS SCANNED, THE TASK IS FOUND TO BE
- / RUNNABLE AND IN CORE, SO IT CAN BE STARTED.

/THE QUESTION IS "HOW CAN TASKS COMMUNICATE?" I.E., "HOW CAN THEY

/ADDRESS MESSAGES TO EACH OTHER?". THE STSK# OR THE TASKS TCB

/CANNOT BE USED TO IDENTIFY A TSK, AS THEY ARE UNKNOWN AT

/ASSEMBLYTIME. THE ONLY THING ABOUT OTHER TSKS THAT IS KNOWN AT

/ASSEMBLYTIME IS THEIR NAME. HOWEVER WE WOULD NOT LIKE THE

/SYSTEM TO SEARCH DOWN A LONG LIST OF NAMES (PREFERABLY STORED ON

/DISK) FOR EACH MESSAGE THAT IS SENT.

/THE PROBLEM IS SOLVED BY INSERTING THE STATIC TSK NUMBER IN THE

/CODE ON THE SPOT WHERE - AT ASSEMBLYTIME - THE NAME WAS WRITTEN,

/THIS IS DONE BY THE TASK LOADER,

/TO BE ABLE TO DO SO, THE TSK LOADER MUST KNOW AHEAD TO WHICH

/TSKS A TSK WILL REFER. THEREFORE TASKS ARE GROUPED IN FAMILIES

/OF TASKS THAT REFER TO ONE ANOTHER.

/"LOADING" A FAMILY:

- /A, FOR EACH MEMBER OF THE FAMILY AN ENTRY IN THE STATIC TASK
- / LIST (STSK#) IS REQUESTED.
- /B, ONE BY ONE EACH MEMBER IS SWAPPED IN, ITS ENTRY IN THE
- / STATIC LIST IS FILLED WITH APPROPRIATE INFO, NAMES
- / OF OTHER TASKS IN ITS CODE ARE REPLACED BY THE

/ CORRESPONDING STATIC TASK NUMBERS, ITS UPDATED CODE IS
/ RESTORED ON DISK
/"UNLOADING" A FAMILY OF TASKS:
/A, STOP ALL MEMBERS (THEY SHOULD BE STOPPED, IF NOT, THIS
/ INDICATES SOME ERROR).
/B, REMOVE THEIR CODE FROM CORE (IF NECESSARY).
/C, CLEAR THEIR ENTRIES IN THE STATIC TASKLIST.

/AS LONG AS A FAMILY IS NOT YET LOADED, NO ONE CAN REFER TO ONE
/OF ITS TASKS, ONCE IT IS LOADED ALL ITS MEMBERS ARE AVAILABLE
/(THOUGH NOT NECESSARILY IN CORE).
/AFTER UNLOADING A FAMILY, ALL ITS MEMBERS ARE COMPLETELY
/DISMISSED FROM CORE, SO THAT NO SPURIOUS MESSAGES WILL ARRIVE AT
/OTHER TASKS THAT LATER USE THE SAME STATIC TASK NUMBERS.

/TO FACILITATE UPDATING THE CODE A SOCALLED TASK INFO BLOCK (TIB)
/PRECEDES THE TASKS CODE ON DISK, THIS BLOCK HOLDS PTRS TO ALL
/PLACES (MAX 83) IN THE CODE WHERE A NAME MUST BE REPLACED BY A
/STSK#. WHEN THE TASK IS ASSEMBLED WITH MC8PAL TASKNAMES ARE
/INDICATED BY USING THE GLTASK OR LCTASK PSEUDO OPS, THESE
/PSEUDO OPS RESERVE ONE LOC AT THE PLACE WHERE THEY ARE USED, AND
/OUTPUT SOME MISTIC BINARY CODE, SO THAT AN APPROPRIATE ENTRY IN
/THE TIB CAN BE BUILT WHEN THE TASK IS INSERTED IN THE TASKLIBRARY.

/LAYOUT OF TIB:

/W0: B0-2 TASKS PRIORITY
/ B3 ZRE@BIT (TASK NEEDS ONE TASK ONLY PART OF PAGE 0),
/ B5 TASK MUST BE LOADED IN CORERESIDENT FIELD
/ B7-11TASKS LENGTH IN PAGES,
/W1 B0 AUTOSTARTBIT. IF THIS BIT IS SET, MCTSLD SENDS A MS
/ TO THAT TASK TO START IT UP.
/W2-4 FAMILY NAME
/W5-6 TASK NAME
/W7-255 3-WORD REFERENCE ENTRIES.
/LAYOUT OF A REFERENCE ENTRY:
/W0-1 TASK NAME (TASKS NAME TO BE REPLACED)
/ W0B0=0: TASK OF GLOBAL FAMILY,
/ W0B0=1: TASK OF OWN FAMILY.
/W2 LOCATION WHERE THE STSK# MUST BE INSERTED.

/IN PRINCIPLE SOME TASKS CAN BE MEMBERS OF DIFFERENT FAMILIES,
/MOREOVER THERE IS ONE FAMILY (NAMED GLOBAL) WHOSE TASKS CAN BE
/ADDRESSED FROM ANY OTHER FAMILY. THE FAMILY GLOBAL IS LOADED
/WHEN THE MCB SYSTEM IS INITIALISED AND IS NEVER UNLOADED, SO ITS
/MEMBERS ARE ALWAYS ADDRESSABLE.

/ORGANISATION OF THE TASK LIBRARY.

/TO MINIMIZE SWAPPING, THE COMPLETE DIRECTORY OF THE TASK LIBRARY
/IS STORED IN THE CODE OF TSLD.
/THIS CAUSES THE COMPLETE DIRECTORY TO BE IN CORE WHEN TSLD IS
/RUN. EACH FAMILY HAS A FAMILY DECLARER (FDC), IT IS ALLOCATED IN
/ONE PAGE.

/LAYOUT OF A FDC:

/W0 0 (TO KEEP THE RELOCATOR OFF).

/W1 NUMBER OF TASKS IN THE FAMILY.

/W2-3 UNUSED.

/W4-127 4-WORD TASK ENTRIES.

/LAYOUT OF TASK ENTRY IN FDC:

/FREE (UNUSED) ENTRIES HAVE ALL WORDS 0.

/USED ENTRIES:

/W0-1 TASK NAME.

/ TASK NAMES CONSIST OF UPTO 4 ALFANUMERIC SYMBOLS, THE
/ FIRST OF WHICH MUST BE A LETTER. THEY ARE STORED IN 6-BIT
/ TRIMMED ASCII, PADDED WITH ZEROES.

/W2 B0 AUTOSTARTBIT.

/ IF SET THE LOADER WILL SEND A MS TO START THAT
/ TSK.

/W2 B1-11 TASKS BLOCKNUMBER ON DISK.

/ IF W2=0, THE TASK IS ALWAYS KNOWN BY THE SYSTEM, AND W3
/ HOLDS THE STATIC TASK NUMBER.

/W3 STSK# IF THE TASK IS LOADED, 0 OTHERWISE.

/NOTE: MAX 31 TASK ENTRIES CAN BE ALLOCATED IN A FDC, SO A
/FAMILY HAS MAX 31 MEMBERS.

/ALL FAMILY DECLARERS ARE DENOTED IN ONE DIRECTORY: THE DIRECTORY
/OF FAMILY DECLARERS (DCDIR).

/LAYOUT OF DCDIR:

/W0 PTR TO START OF TSLD CODE.

/W1 0 (TO KEEP OFF THE RELOCATOR)

/W2 JMP 1, -2. THIS INSTRUCTION WILL START UP TSLD.

/W3 NUMBER OF PAGES AVAILABLE FOR FDCS.

/W4-127 FDC ENTRIES.

/LAYOUT OF FDC ENTRY IN DCDIR:

/FREE (UNUSED) ENTRIES HAVE ALL WORDS 0.

/USED ENTRIES:

/W0-2 FAMILY NAME.

/ FAMILY NAMES CONSIST OF UPTO 6 ALFANUMERIC SYMBOLS, THE
/ FIRST OF WHICH IS A LETTER. THESE SYMBOLS ARE STORED IN
/ 6-BIT TRIMMED ASCII, PADDED WITH ZEROES.

/W3 B0: LOADBIT.

/ 1: FAMILY IS LOADED, 0: FAMILY IS UNLOADED.

/ B1-5: PAGENUMBER OF FAMILY DECLARER.

/ B6-11: MUST BE 0.

/TASK STORAGE ON DISK.

/

/A TASK IS STORED IN A NUMBER OF CONSECUTIVE BLOCKS ON DISK
/((LOGICAL UNIT 0). THE FIRST IS THE TIB.

/REFERENCES TO A BLOCKNUMBER OF A TASK ARE ALWAYS GIVEN WITH AN
/OFFSET OF TSBLOK (A SYSTEM CONSTANT).

/BLOCKNUMBERS OF TASKS MUST BE POSITIVE I.E. MAX 2047.
/THE FIRST BLOCK (TSBLOK) OF THE TASK STORAGE AREA IS USED AS A
/BITMAP OF OCCUPIED BLOCKS. IF A BLOCK IS USED FOR TASK STORAGE
/THE CORRESPONDING BIT IS SET. WE USE B4-11 OF EACH WORD.
/SO W0B11 CORESSPONDS TO TSBLOK, B10 TO TSBLOK+1...
/W1 B11 TO TSBLOK+10, ETC.
/THE FIRST TASK MUST ALWAYS BE "GLOBAL.TSLD", AND IT MUST BE
/ALLOCATED FROM TSBLOK UPWARD.
/SO ITS TIB IS STORED ON TSBLOK+0 AND ITS CODE FROM TSBLOK+1
/UPWARD. AS THE DIRECTORIES OF THE TASK LIBRARY ARE STORED AS
/PART OF THE CODE OF "GLOBAL.TSLD" THEY MAY BE FOUND AS FOLLOWS:
/BITMAP AT TSBLOK+0. (NO ACTUAL TIB OF TSLD).
/DCDIR AT TSBLOK+1, FIRST PAGE.
/NRT AT TSBLOK+1, SECOND PAGE.
/FDCS AT TSBLOK+2, SSQQ.

/THE NAME REFERENCE TABLE (NRT).

/

/IT IS CONVENIENT, THOUGH NOT NECESSARY, TO KEEP A LIST OF TASKS
/THAT ARE LOADED. THIS LIST IS CALLED THE NAME REFERENCE TABLE,
/AND IT IS KEPT ON THE SECOND PAGE OF TSBLOK+1.
/W0 OF THE NRT MUST 0, TO KEEP THE RELOACTOR OFF.
/ALL OTHER WORDS CORRESPOND TO ENTRIES IN THE STL IN AN EVIDENT
/WAY.

/LAYOUT OF AN ENTRY IN THE NRT:

/0 NO TASK IS LOADED IN THE CORRESPONDING ENTRY OF THE STL
/ (WARNING: TSLD DID NOT USE THAT ENTRY, BUT PERHAPS OTHERS
/ DID!)

/B1-5 NUMBER OF FDC IN DCDIR.
/B7-11 NUMBER OF TASK ENTRY IN FDC.

/DIRECTORY OF FAMILY DECLARERS.

/NOTE: USING PTRS IS OK, AS TSLD IS 37 PAGES, SO THAT
/NO ACTUAL RELOCATION CAN OCCUR.

00200	5205	DCDIR, START	/RELOCATED PTR TO START.
00201	0000	0	/END OF RELOCATED CODE.
00202	5600	JMP I DCDIR	
00203	0021	FDCMAX-GLOBAL#200	/NUMBER OF PAGES AVAILABLE FOR
			/FDCS.
00204	0714	0714;1702;0114;GLOBAL#2	
00205	1702		
00206	0114		
00207	0300		
00210	0000	ZBLOCK 200-..177	

/NAME REFERENCE TABLE.
/EACH ENTRY CORRESPONDS TO AN ENTRY IN THE STATIC TASK
/LIST (NRT[1] TO STL[0], NRT[2] TO STL[1] ...).
/NRT[0] MUST BE 0 TO KEEP THE RELOCATOR OFF.

IFNZRO 176-MAXSTL6^4000 <ASS ERROR!>
/THE NRT CLEARLY HAS ONLY 176 ENTRIES.

00400 0000 /INITIALLY CLEARED.
NRT, ZBLOCK 200

/FAMILY DECLARERS.
/EACH FAMILY DECLARER OCCUPIES 1 PAGE OF CODE.

/INITIAL GLOBAL DECLARER,
GLOBAL,

00600 0000 0;6;0;0

00601 0006

00602 0000

00603 0000

00604 1104

DEVICE IDLE;0;SIDLE

00605 1405

00606 0000

00607 0000

00610 2327

DEVICE SWAP;0;SMTSWAP

00611 0120

00612 0000

00613 0001

00614 0624

DEVICE FTCH;0;SMTFTCH

00615 0310

00616 0000

00617 0002

00620 2411

DEVICE TIME;0;SMTTIME

00621 1505

00622 0000

00623 0003

00624 2304

DEVICE SDSK;0;SMTSDSK

00625 2313

00626 0000

00627 0004

00630 2423

DEVICE TS LD;0;XTSLD, 0/SMTTSLD

00631 1404

00632 0000

00633 0000

00634 0000

ZBLOCK 200-..177 /ALL OTHERS UNUSED.

/WHEN BUILDING THE SYSTEM, HERE IS THE INITIAL BITMAP.
BITMAP, 377;377;1 /TSBLOK+16 BLOCKS OF TS LD.

01000 0377

01001 0377

01002 0001

/ALL OTHER FDCS INITIALLY CLEARED.
ZBLOCK FDCMAX=.

01003 0000

5000 PAGE

/FORCE RELOCATION TO NEXT PAGE=5000.

5000 /BUFFER FOR TASK INFO BLOK.
TIBBUF=,^7600

/IN THIS FIRST PAGE OF TIBBUF GOES THE CODE TO BUILD
/INITIAL BITMAP AND DIRECTORIES.
/ONE FAMILY DECLARER FOR GLOBAL IS SET UP. ALL OTHERS
/ARE ZERO.
/THE WHOLE TSLD IS WRITTEN ON THE DISK FROM TSBLOK+1.

05000 0000 0 /TO SATISFY FUTURE RELOCATION.

/NOTE: THIS CODE IS ONLY EXECUTED WHEN BUILDING A
/COMPLETELY FRESH TASK LIBRARY. THIS CODE IS NOT INTENDED
/TO BE EXECUTED AS TASKCODE, SO IT IS NOT RELOCATED.
MCTSLD,
/FIRST THE INITIAL BITMAP MUST BE WRITTEN ON DISK, ALLOCATED ON
/LOGICAL UNIT 0 AT TSBLOK.
/NEXT THIS COMPLETE FLD MINUS PAGE 0 MUST BE WRITTEN ON DISK,
/IT MUST BE ALLOCATED ON THE FUTURE MC8 SYSTEM DISK
/LOGICAL UNIT 0, FROM TSBLOK+1.
/THE WAY TO ACCOMPLISH THIS DEPENDS ON THE SYSTEM WHICH
/IS CURRENTLY RUNNING.
/BELOW FOLLOWS A VERSION, ASSUMING THAT THE MC8 SYSTEM DISK
/LU 0 BLOCK 0 CORRESPONDS TO THE OS/8 SYSTEMDEVICE
/BLOCK 0.

0020 CURFLD=20 /WE ARE RUNNING IN FLD 2.
05001 6202 CIF 0
05002 4777 JMS I (7607
05003 4220 4200+CURFLD/WRITE TWO PAGES,
05004 1000 BITMAP
05005 6000 TSBLOK
05006 7402 HLT
05007 3776 DCA I (BITMAP /BITMAP WAS STORED IN SOME FDC, AND
05010 3775 DCA I (BITMAP+1/HENCE MUST BE CLEREARED BE FORE THE
05011 3774 DCA I (BITMAP+2/DIRECTORIES ARE WRITTEN OUT.

05012 6202 CIF 0
05013 4777 JMS I (7607
05014 7720 7700+CURFLD
05015 0200 200
05016 6001 TSBLOK+1
05017 7402 HLT
05020 6203 CIF CDF 0
05021 5773 JMP I (7600

05173 7600
05174 1002
05175 1001
05176 1000
05177 7607
5204 PAGE

/INITIALISATION CODE IN SECOND PAGE OF TIB BUFFER.
/CODE TO INITIALISE DIRECTORIES AND TO LOAD GLOBAL.

05204 0000 0 /KEEP OFF RELOCATOR.

05205 3157 START, DCA MSP /STORE PTR TO MS.

/IF A MS IS RECEIVED TO LOAD GLOBAL, WE ASSUME THAT THE
/SYSTEM MUST BE REINITIALISED. OTHER MSS ARE PASSED
/TO MSGIN.

05206 1157 TAD MSP
05207 3110 DCA X
05210 6201 CDF 0
05211 1510 TAD I X
05212 1377 TAD (-0714 /GL
05213 7640 SZA CLA
05214 5600 JMP I)MSGIN
05215 2110 ISZ X
05216 1510 TAD I X
05217 1376 TAD (-1702 /OB
05220 7640 SZA CLA
05221 5600 JMP I)MSGIN
05222 2110 ISZ X
05223 1510 TAD I X
05224 1375 TAD (-0114 /AL
05225 7640 SZA CLA
05226 5600 JMP I)MSGIN
05227 4051 TSLIN, CDFCUR
05230 7307 AC4
05231 1201 TAD)DCDIR
05232 3150 DCA DCPTR /PTR IN DECLARER DIRECTORY.
05233 1374 TAD (-37
05234 3151 DCA DCCTR /AT MOST 37OCT DECLARERS.
/CLEAR THE LOADBITS OF ALL FAMILIES.
05235 7325 DCINIT, AC3
05236 1150 TAD DCPTR
05237 3150 DCA DCPTR /SKIP OVER THE FAMILYNAME.
05240 7350 AC3777
05241 0550 AND I DCPTR
05242 7450 SNA /IS THIS ENTRY USED?
05243 5267 JMP DCIN2 /NO. SKIP INIT OF FDC.
05244 3550 DCA I DCPTR /LOADBIT CLEARED.

05245 1550 TAD I DCPTR
05246 7104 CLL RAL
05247 0146 AND C7600
05250 1122 TAD C4
05251 3152 DCA FDCPTR
05252 1374 TAD (-37
05253 3153 DCA FDCCTR

/CLEAR ALL STATIC TASK NUMBERS IN FAMILY DECLARERS.
/NOTE: IF NO DISKBLOCK IS INDICATED FOR A GIVEN TASK,
/THEN THAT TASK IS ALWAYS PRESENT IN THE SYSTEM,
/AND SO ITS STSK# MUST NOT BE CLEARED.

05254 7326 FDINIT, AC2

```

05255 1152      TAD      FDCPTR
05256 3152      DCA      FDCPTR /SKIP OVER TASK NAME.
05257 7350      AC3777
05260 0552      AND I    FDCPTR /GET BLOCK#; 0 INDICATES TSK ALWAYS IN
05261 2152      ISZ      FDCPTR /SYSTEM.
05262 7640      SZA CLA
05263 3552      DCA I    FDCPTR /CLEAR STSK#
05264 2152      ISZ      FDCPTR
05265 2153      ISZ      FDCPTR
05266 5254      JMP      FDNIT

05267 2150      DCIN2,   ISZ      DCPTR
05270 2151      ISZ      DCCTR /ALL DECLARERS INITIALISED?
05271 5235      JMP      DCINIT

```

/CLEAR THE NAME REFERENCE TABLE.

```

05272 1202      TAD      )NRT
05273 3150      DCA      DCPTR /JUST A PTR.
05274 1373      TAD      (=MAXSTL-1
05275 3151      DCA      DCCTR /LENGTH OF STATIC TASK LIST +1.
05276 3550      DCA I    DCPTR
05277 2150      ISZ      DCPTR
05300 2151      ISZ      DCCTR
05301 5276      JMP      .-3

```

/WE STILL DONT KNOW OUR OWN STSK#.

/IT WILL BE PASSED IN THE NEXT MS.

```

05302 4054      CALLJ    WTMS+KEEP
05303 1014
05304 4014      MSFREE
05305 3603      DCA I    )XTSLD

```

/ALL INITIALISATION DONE: SIMULATE A NORMAL CALL.

```

05306 5600      JMP I    )MSGIN

```

```

05200 5404
05201 0200
05202 0400
05203 0633
05373 7700
05374 7741
05375 7664
05376 6076
05377 7064
5403 PAGE

```

0

MSGIN, CDF 0

Line	Address	Instruction	Comment
05404	6201	MSGIN, CDF 0	
05405	1157	TAD MSP	/STORE MSPTR FOR MSREPORT
05406	3301	DCA MSPREP	
05407	3161	DCA SWFLD	/NO FLD CLAIMED YET,
05410	3162	DCA SWMSP	/NO MS CLAIMED YET,
05411	1157	TAD MSP	
05412	3110	DCA X	
05413	7350	AC3777	
05414	0510	AND 1 X	/GET FAMILY NAME OUT OF MS,
05415	7041	CIA	
05416	3154	DCA FNAME	
05417	2110	ISZ X	
05420	1510	TAD 1 X	
05421	7041	CIA	
05422	3155	DCA FNAME+1	
05423	2110	ISZ X	
05424	1510	TAD 1 X	
05425	7041	CIA	
05426	3156	DCA FNAME+2	
05427	4051	CDFCUR	
05430	7307	AC4	
05431	1200	TAD)DCDIR	
05432	3150	DCA DCPTR	/SEARCH DCDIR FOR A DECLARER OF THIS
05433	1377	TAD (-37	/NAME,
05434	3151	DCA DCCTR	
05435	1550	SDCLP, TAD 1 DCPTR	
05436	2150	ISZ DCPTR	
05437	7450	SNA	/ANYTHING IN THIS ENTRY?
05440	5257	JMP SDCSKP	/NO,
05441	1154	TAD FNAME	
05442	7640	SZA CLA	
05443	5257	JMP SDCSKP	
05444	1550	TAD 1 DCPTR	
05445	2150	ISZ DCPTR	
05446	1155	TAD FNAME+1	
05447	7640	SZA CLA	
05450	5260	JMP SDCSKP+1	
05451	1550	TAD 1 DCPTR	
05452	2150	ISZ DCPTR	
05453	1156	TAD FNAME+2	
05454	7640	SZA CLA	
05455	5261	JMP SDCSKP+2	
05456	5304	JMP SDCFND	
05457	2150	SDCSKP, ISZ DCPTR	
05460	2150	ISZ DCPTR	
05461	2150	ISZ DCPTR	
05462	2151	ISZ DCCTR	
05463	5235	JMP SDCLP	
05464	7201	ER1, CLA IAC	/SIGNAL ERROR CONDITION 1, /INCORRECT FAMILY NAME.

```

/END OF THIS RUN,
05465 6201 DONE, CDF 0
05466 3557 DCA I MSP /COMPLETION OR ERROR CONDITION IN MS,
05467 1162 TAD SWMSP /ANY MS CLAIMED?
05470 7440 SZA
05471 4014 MSFREE /YES, FREE IT.
05472 7200 CLA
05473 1161 TAD SWFLD /ANY FLD CLAIMED
05474 7440 SZA
05475 4054 CALL; RTNFLD /YES RETURN IT, OR: INNOCENT AND.
05476 0036
05477 4054 CALL; RP
05500 0013
05501 0000 MSPREP, 0
05502 4054 CALL; EXIT
05503 0054

```

```

/WE FOUND THE SPECIFIED FAMILY DECLARER.
/DCPTR PTS AT ITS ENTRY W3.
05504 1550 SDCFND, TAD I DCPTR /SET FSTRT TO FDC+4 (FIRST 4 LOCS
05505 7104 CLL RAL /UNUSED).
05506 0146 AND C7600
05507 1122 TAD C4
05510 3160 DCA FSTRT
05511 6201 CDF 0
05512 1557 TAD I MSP
05513 4051 CDFCUR
05514 7700 SMA CLA /LOAD OR UNLOAD FAMILY?
05515 5601 JMP I )LOAD
05516 5602 JMP I )UNLOAD

```

```

/INITIALISE FOR SEARCHING ONE FDC.
/FSTRT=FDC+4.
05517 0000 FDCSET, 0
05520 1160 TAD FSTRT /PTRS.
05521 3152 DCA FDCPTR
05522 1377 TAD (-37 /MAX 37OCT TSKS IN FDC,
05523 3153 DCA FDCCTR
05524 5717 JMP I FDCSET
05400 0200
05401 5611
05402 6307
05577 7741
5611 PAGE

```

/LOAD A FAMILY OF TASKS.
 /MINUS ITS NAME IS DENOTED IN FNAME,FNAME+1,FNAME+2.
 /DCPTR PTS AT ITS ENTRY IN DCDIR W3.

```

05611 1550 LOAD, TAD I DCPTR /GET ENTRY[3]; NEGATIVE MEANS ALREADY
05612 7500 SMA /LOADED.
05613 5216 JMP .+3
05614 7326 ER2, AC2 /ERROR CONDITION 2: ALREADY LOADED.
05615 5600 JMP I )DONE
05616 4054 CALL; REQFLD /REQUEST FLD TO STORE TASKS CODE.
05617 0034
05620 4054 CALL; REQCDF /SET UP FOR MINIMUMTIME CDF ROUTINE.
05621 0052
05622 3161 DCA SWFLD
05623 4601 JMS I )FDCSET /SET UP FOR SEARCHING ONE FDC.

```

/LOOP TO REQUEST ENTRIES IN STL.

```

05624 7326 LD1LP, AC2
05625 1152 TAD FDCPTR
05626 3152 DCA FDCPTR /SKIP OVER TASK NAME
05627 7350 AC3777
05630 0552 AND I FDCPTR /FETCH PTR TO TIB OF TASK
05631 2152 ISZ FDCPTR
05632 7450 SNA /0 INDICATES TASK ALREADY KNOWN
/IN SYSTEM OR FREE ENTRY.

05633 5243 JMP LD1SKP
05634 7001 IAC /PTR TO TIB+1 (FIRST CODE BLOCK)
05635 3240 DCA RQARG /IN ARG1.
05636 4054 CALL; REQSTL
05637 0041
05640 0000 RQARG, 0;0
05641 0000
05642 3552 DCA I FDCPTR /STORE STSK# RETURNED IN AC IN FDC.
05643 2152 LD1SKP, ISZ FDCPTR
05644 2153 ISZ FDCPTR
05645 5224 JMP LD1LP

05646 4601 JMS I )FDCSET /AGAIN LOOKING AT ALL TASKS.
05647 4014 MSREQ /REQUEST MS FOR SWAPPING.
05650 3162 DCA SWMSP

```

/LOOP TO UPDATE THE CODE OF EACH TASK AND TO SET STL[TASK,1].

```

05651 7326 LD2LP, AC2
05652 1152 TAD FDCPTR
05653 3152 DCA FDCPTR /SKIP OVER TSKNAME.
05654 7350 AC3777
05655 0552 AND I FDCPTR /GET BLOKNUMBER
05656 2152 ISZ FDCPTR
05657 7450 SNA /0 INDICATES TASK ALREADY IN SYSTEM
/OR FREE ENTRY.

05660 5602 JMP I )LD2SKP
05661 1377 TAD (TSBLOK
05662 3163 DCA TBLOK /BLOCKNUMBER OF TASK.
05663 1162 TAD SWMSP

```

```

05664 3303      DCA      MSP0
05665 1162      TAD      SWMSP      /SEND MS TO FETCH TIB,
05666 3110      DCA      X          /PTR IN MS.
05667 6201      CDF 0
05670 1133      TAD      C200      /READ TWO PAGES FROM LOGICAL UNIT 0.
05671 3510      DCA I      X
05672 2110      ISZ      X
05673 1203      TAD      )TIBBUF
05674 3510      DCA I      X
05675 2110      ISZ      X
05676 1163      TAD      TBLOK      /BLOCKNUMBER
05677 3510      DCA I      X
05700 4051      CDFCUR
05701 4054      CALLJ      SNDWTR+DFPARM
05702 0107
05703 0000      MSP0,      0
05704 0004      SMTSDSK
05705 3334      DCA      MSP1      /SHIFT THIS USEFUL PTR AHEAD,
05706 6201      CDF 0          /CHECK REPORTED MS FOR I/O ERROR
05707 1703      TAD I      MSP0
05710 4051      CDFCUR
05711 7640      SZA CLA          /NONZERO INDICATES ERROR.
05712 5604      JMP I      )LDIOER
      /SEND MS TO SWAP TASKS CODE IN.
05713 1603      TAD I      )TIBBUF /GET LENGTH
05714 0125      AND      C37
05715 7106      CLL RTL;RTL;RTL
05716 7006
05717 7006
05720 6201      CDF 0
05721 3703      DCA I      MSP0      /NUMBER OF PAGES FROM UNIT 0.
05722 2303      ISZ      MSP0
05723 1133      TAD      C200
05724 3703      DCA I      MSP0      /CODE BUFFER STARTS AT 200.
05725 2303      ISZ      MSP0
05726 2163      ISZ      TBLOK      /PTR TO FIRST CODE BLOCK.
05727 1163      TAD      TBLOK      /FIRST CODE BLOCK.
05730 3703      DCA I      MSP0      /BLOCKNUMBER OF TASKS CODE.
05731 4020      VRCDF      /DF TO TASK CODE BUFFER FIELD
05732 4054      CALLJ      SNDMS+DFPARM
05733 0103
05734 0000      MSP1,      0
05735 0004      SMTSDSK /SWAP TASK CODE IN.
05736 4051      CDFCUR
05737 1334      TAD      MSP1
05740 3605      DCA I      )MSP2      /SHIFT PTR AHEAD.
05741 1552      TAD I      FDCPTR      /UPDATE STATIC TASK LIST.
05742 0132      AND      C177
05743 7104      CLL RAL
05744 1376      TAD      (STLST
05745 3010      DCA      X0          /PTR TO STL[TASK,1]
05746 1603      TAD I      )TIBBUF      /LENGTH+CRES+ZREQ+PRIO
05747 6201      CDF 0
05750 3410      DCA I      X0
05751 4051      CDFCUR

```

/REPLACE THE TASKNAMES IN THE 3-WORD ENTRIES IN THE TIB BY THE
/CORRESPONDING STATIC TASKNUMBERS.

05752	1206	TAD	)TIBBUF+7
05753	3164	DCA	TIBPTR
05754	1375	TAD	(-123 /NUMBER OF ENTRIES.
05755	3151	DCA	DCCTR /JUST A CTR HANGING AROUND
05756	4607	JMS I	)TNSET /REPLACE.
05757	5362	JMP	.,+3 /RTN1: ALL ENTRIES DONE
05760	2151	ISZ	DCCTR /MORE ENTRIES TO DO?
05761	5356	JMP	.,-3 /YES.

/ALL ENTRIES IN TIB DONE.
/PREPARE FOR UPDATING CODE.

05762	1206	TAD	)TIBBUF+7
05763	3164	DCA	TIBPTR
05764	1375	TAD	(-123 /NUMBER OF ENTRIES IN TIB
05765	3151	DCA	DCCTR
05766	5610	JMP I	)LD2LP2

05600 5465
05601 5517
05602 6057
05603 5000
05604 6305
05605 6013
05606 5007
05607 6204
05610 6011
05775 7655
05776 4400
05777 6000
6010 PAGE

```

06010 0000 FDCNR, 0

06011 4054 LD2LP2, CALL; WTRP /WAIT FOR CODE SWAP IN COMPLETED.
06012 0005
06013 0000 MSP2, 0
06014 3247 DCA MSP3 /KEEP SHIFTING THIS PTR AHEAD.
06015 6201 CDF 0
06016 1613 TAD I MSP2 /I/O ERROR?
06017 4051 CDFCUR
06020 7640 SZA CLA
06021 5600 JMP I )LDIOER

/UPDATE TASKS CODE.
06022 4601 JMS I )TSKUPD /UPDATE LOC INDICATED BY TIBENTRY.
06023 5226 JMP .+3 /RTN1: ALL ENTRIES DONE
06024 2151 ISZ DCCTR /RTN2: PERHAPS MORE TO DO
06025 5222 JMP .-3

/CODE UPDATED. SEND MS TO SWAP TASK OUT AGAIN.
06026 1602 TAD I )TIBBUF /GET LENGTH
06027 0125 AND C37
06030 7106 CLL RTL;RTL;RTL
06031 7006
06032 7006
06033 1377 TAD (4000
06034 6201 CDF 0
06035 3613 DCA I MSP2 /WRITE N PAGES ON LOGICAL UNIT 0.
06036 2213 ISZ MSP2
06037 1133 TAD C200 /TASK CODE BUFFER STARTS AT 200
06040 3613 DCA I MSP2 /CORE ADDRESS
06041 2213 ISZ MSP2
06042 1163 TAD TBLOK /BLOCKNUMBER
06043 3613 DCA I MSP2
06044 4020 VRCDF /DF TO TASK CODE BUFFER FIELD
06045 4054 CALL; SNDWTR+DFPARM
06046 0107
06047 0000 MSP3, 0
06050 0004 SMTSDSK
06051 3360 DCA MSP4 /STILL SHIFTING AHEAD.
06052 6201 CDF 0
06053 1647 TAD I MSP3
06054 4051 CDFCUR
06055 7640 SZA CLA
06056 5600 JMP I )LDIOER

06057 2152 LD2SKP, ISZ FDCPTR /NEXT TASK
06060 2153 ISZ FDCCTR
06061 5603 JMP I )LD2LP

/ACTUAL LOADING COMPLETED. SET LOADED BIT.
06062 7350 AC3777
06063 0550 AND I DCPTR
06064 1377 TAD (4000 /SET LOADED BIT
06065 3550 DCA I DCPTR

```

/UPDATE NAME REFERENCE TABLE.
 /EACH ENTRY IN THE NRT CORRESPONDS TO ONE ENTRY IN THE STATIC TASK
 /LIST. B1-5 CONTAIN THE FAMILY DECLARER NUMBER, B7-11 CONTAIN THE

06066	7346	ACM3		/TASKNUMBER, COMPUTE FDCNUMBER,
06067	1150	TAD	DCPTR	
06070	0132	AND	C177	/FDCNUMBER+4
06071	7106	CLL	RTL;RTL	
06072	7006			
06073	3210	DCA	FDCNR	/FDCNUMBER IN APPROPRIATE BITS.
06074	4604	JMS	I)FDCSET	/INITIALISE FOR READING
				/OUT ALL STATIC TASKNUMBERS.

/LOOP TO UPDATE NRT AND START UP TASKS IF THE AUTOSTARTBIT IS
 /SET.

06075	1552	LD3LP,	TAD I	FDCPTR	/0 INDICATES EMPTY ENTRY
06076	2152		ISZ	FDCPTR	
06077	2152		ISZ	FDCPTR	
06100	7650		SNA	CLA	
06101	5334		JMP	LD3SKP	
06102	1552		TAD I	FDCPTR	
06103	2152		ISZ	FDCPTR	
06104	7700		SMA	CLA	/AUTOSTARTBIT?
06105	5321		JMP	LD3LP2	/NO
06106	4014		MSREQ		/YES; SEND MS TO THIS TASK.
06107	3317		DCA	LD3MS	
06110	1552		TAD I	FDCPTR	
06111	3320		DCA	LD3MS+1	
06112	6201		CDF	0	
06113	3717		DCA I	LD3MS	
06114	4051		CDFCUR		
06115	4054		CALL;	SNDMS+NONREP	
06116	2003				
06117	0000	LD3MS,		0;0	
06120	0000				
06121	1552	LD3LP2,	TAD I	FDCPTR	/GET STSK#
06122	0132		AND	C177	
06123	1205		TAD	)NRT+1	/ADD NRT OFFSET
06124	3150		DCA	DCPTR	/PTR HANGING AROUND
06125	7346		ACM3		/COMPUTE TASKNUMBER
06126	1152		TAD	FDCPTR	
06127	0132		AND	C177	/TASKNUMBER *4
06130	7112		CLL	RTR	
06131	1210		TAD	FDCNR	/ADD DECLARERNUMBER
06132	3550		DCA I	DCPTR	/AND STORE IN NRT.
06133	7410		SKP		
06134	2152	LD3SKP,	ISZ	FDCPTR	
06135	2152		ISZ	FDCPTR	
06136	2153		ISZ	FDCCTR	
06137	5275		JMP	LD3LP	

/EVERYTHING IS OK NOW.
 /REWRITE TASKDIRECTORIES ON DISK.

06140	1162	TAD	SWMSP	/SET UP PTRS TO SWAPMESSAGE,
06141	3360	DCA	MSP4	

```
06142 1162          TAD      SWMSP  /IT IS NOT SURE THAT THEY ARE SET
06143 3110          DCA      X      /CORRECT.
06144 6201          CDF 0
          DLENG=FDCMAX-DCDIR/2+100^3600
06145 1376          TAD      (DLENG+4000 /WRITE OUT DLENG PAGES.
06146 3510          DCA I      X
06147 2110          ISZ      X
06150 1206          TAD      )DCDIR
06151 3510          DCA I      X      /CORE ADDRESS.
06152 2110          ISZ      X
06153 1375          TAD      (TSBLOK+1/STARTING BLOCK OF DIRECTORIES
06154 3510          DCA I      X
06155 4051          CDFCUR
06156 4054          CALL;     SNDWTR+DFPARM
06157 0107
06160 0000          MSP4,      0
06161 0004          SMTSDSK
06162 4014          MSFREE    /FREE MS AND CHECK I/O STATUS
06163 3210          DCA      FDCNR  /JUST A TEMP.
06164 3162          DCA      SWMSP  /WE FREED THE MS.
06165 1210          TAD      FDCNR
06166 7440          SZA
06167 7307          AC4
06170 5607          JMP I      )DONE      /ERROR 4: DIRECTORY WRITE OUT ERROR.

06000 6305
06001 6264
06002 5000
06003 5651
06004 5517
06005 0401
06006 0200
06007 5465
06175 6001
06176 6400
06177 4000
        6201  PAGE
```

06201 0604 GLSTRT, GLOBAL+4
 06202 0000 TNPTR, 0
 06203 0000 TNCTR, 0

/SOME USEFUL ROUTINES FOR THE LOADER.

/REPLACE TASKNAME PTD AT BY TIBPTR BY THE CORRESPONDING STSK#.
 /NOTE: THE GLOBAL DECLARER STARTS AT GLOBAL, THE START OF THE
 /FAMILY DECLARER IS COMPUTED USING DCPTR.

06204 0000 TNSET, 0
 06205 1564 TAD I TIBPTR /MORE ENTRIES TO DO?
 06206 7450 SNA
 06207 5604 JMP I TNSET /NO.
 06210 7700 SMA CLA
 06211 1160 TAD FSTRT /OWN FAMILY
 06212 7450 SNA
 06213 1201 TAD GLSTRT /GLOBAL FAMILY
 06214 3202 DCA TNPTR
 06215 1377 TAD (-37
 06216 3203 DCA TNCTR
 06217 7350 AC3777
 06220 0564 AND I TIBPTR
 06221 2164 ISZ TIBPTR
 06222 7041 CIA
 06223 3310 DCA TNAME
 06224 1564 TAD I TIBPTR
 06225 7041 CIA
 06226 3311 DCA TNAME+1
 06227 1602 TNSLP, TAD I TNPTR /GET NAME FROM DECLARER
 06230 2202 ISZ TNPTR
 06231 7450 SNA
 06232 5242 JMP TNSKP
 06233 1310 TAD TNAME
 06234 7640 SZA CLA
 06235 5242 JMP TNSKP
 06236 1602 TAD I TNPTR /SECOND WORD OF TASKNAME
 06237 1311 TAD TNAME+1
 06240 7650 SNA CLA
 06241 5254 JMP TNFND
 06242 2202 TNSKP, ISZ TNPTR /NOT THIS TASK OF DECLARER, KEEP
 06243 2202 ISZ TNPTR /SEARCHING.
 06244 2202 ISZ TNPTR
 06245 2203 ISZ TNCTR /DECLARER EXHAUSTED?
 06246 5227 JMP TNSLP
 06247 4054 ER3, CALL
 06250 0005 WTRP
 06251 0000 0 /WAIT FOR REPORT OF SWPMS
 06252 7325 AC3 /WRONG TIB.
 06253 5600 JMP I)DONE
 06254 2202 TNFND, ISZ TNPTR /WE FOUND THE CORRECT TASK.
 06255 2202 ISZ TNPTR /PTR TO ITS STSK#
 06256 1602 TAD I TNPTR /GET IT.
 06257 3564 DCA I TIBPTR /STORE IN TIB. W1 OF THE ENTRY.

```

06260 2164      ISZ      TIBPTR  /UPDATE PTR FOR NEXT ENTRY
06261 2164      ISZ      TIBPTR
06262 2204      ISZ      TNSET   /TAKE SECOND RETURN.
06263 5604      JMP      TNSET

```

```

/UPDATE TASK CODE.
/TIBPTR PTS AT A 3-WORD ENTRY IN TIB.
/W1 HOLDS THE STSK#
/W2 PTS AT THE LOC WHERE IT MUST BE STORED.
TSKUPD, 0

```

```

06264 0000      TAD      TIBPTR  /MORE ENTRIES TO DO?
06265 1564      SNA      CLA
06266 7650      JMP      TSKUPD  /NO, TAKE RETURN1.
06270 2264      ISZ      TSKUPD  /YES, TAKE RETURN 2.
06271 2164      ISZ      TIBPTR
06272 1564      TAD      TIBPTR  /GET STSK#
06273 3203      DCA      TNCTR   /JUST A TEMP.
06274 2164      ISZ      TIBPTR  /PTR TO W2
06275 1564      TAD      TIBPTR  /GET PTR IN TASKS CODE,
06276 2164      ISZ      TIBPTR
06277 3202      DCA      TNPTR   /STORE IN OWN PAGE.
06300 1203      TAD      TNCTR   /REFETCH STSK#
06301 4020      VRCDF
06302 3602      DCA      TNPTR
06303 4051      CDFCUR
06304 5664      JMP      TSKUPD

```

```

/I/O ERROR WHILE LOADING.

```

```

06305 7330      LDIOER, AC4000
06306 5600      JMP      )DONE

```

```

06307 4023      UNLOAD, ERHLT

```

```

06310 0000      TNAME, 0;0
06311 0000

```

```

06200 5465
06377 7741
6400      PAGE

```

```

06400 0000      ZBLOCK 7777-. /FORCE USING ALL PAGES,

```

```
      /ZREQ AREA,  
      NOPUNCH  
00150 0150 *150  
00150 0000 DCPTR, 0  
00151 0000 DCCTR, 0  
00152 0000 FDCPTR, 0  
00153 0000 FDCCTR, 0  
00154 0000 FNAME, 0;0;0  
00155 0000  
00156 0000  
00157 0000 MSP, 0  
00160 0000 FSTRT, 0      /FAMILY DECLARER +4.  
00161 0000 SWFLD, 0  
00162 0000 SWMSP, 0  
00163 0000 TBLOK, 0  
00164 0000 TIBPTR, 0  
  
      ENPUNCH
```

MC8PAL-V1A 02/14/77 PAGE 16

\$

MC8PAL-V1A 02/14/77 PAGE 17

BITMAP	1000	TNSKP	6242
CURFLD	0020	TNSLP	6227
DCCTR	0151	TSKUPD	6264
DCINIT	5235	TSLIN	5227
DCIN2	5267	UNLOAD	6307
DCPTR	0150	XTSLD	0633
DLENG	2400		
DONE	5465		
ER1	5464		
ER2	5614		
ER3	6247		
FDCCTR	0153		
FDCNR	6010		
FDCPTR	0152		
FDCSET	5517		
FDINIT	5254		
FNAME	0154		
FSTRT	0160		
GLOBAL	0600		
GLSTRT	6201		
LDIOER	6305		
LD1LP	5624		
LD1SKP	5643		
LD2LP	5651		
LD2LP2	6011		
LD2SKP	6057		
LD3LP	6075		
LD3LP2	6121		
LD3MS	6117		
LD3SKP	6134		
LOAD	5611		
MCTSLD	5001		
MSGIN	5404		
MSP	0157		
MSPREP	5501		
MSP0	5703		
MSP1	5734		
MSP2	6013		
MSP3	6047		
MSP4	6160		
RQARG	5640		
SDCFND	5504		
SDCLP	5435		
SDCSKP	5457		
START	5205		
SWFLD	0161		
SWMSP	0162		
TBLOK	0163		
TIBBUF	5000		
TIBPTR	0164		
TNAME	6310		
TNCTR	6203		
TNFND	6254		
TNPTR	6202		
TNSET	6204		

ERRORS DETECTED: 0
LINKS GENERATED: 0

APPENDIX D.

THE SYSTEM BULLETIN PRESENTED HEREAFTER WAS PREPARED AFTER
RUNNING THE TASKS RTTY AND WTTY FOR A WHILE.

NOTE: GLOBAL.LDTS IS A TASK THAT WAS RUN TO LOAD THE TASKS
TEST,RTTY AND TEST,WTTY.

MC8 SYSTEM BULLETIN.

DATE: 14-FEB-77
RUN NUMBER: 3

RUNNED DURING: 0 HRS 6 MIN 13.2 SEC

SYSTEM VARIABLES.

CURTSK: 0000
SCDREQ,SCDINH: 0001 0000
INTAC,INTFL,INTPC: 0000 0000 1447
MONAC,MONFL,MONPC: 0000 0000 2352
MONFUNC,MARG1,MARG2: 1014 4273 0010

HPRIO=PRIORITY 0010

RUN QUEUES.

0000	0000	
0001	0000	
0002	0000	
0003	0000	
0004	0000	
0005	0000	
0006	0000	
0007	0000	
0010	5113	0000

SKIP CHAIN.

0207	6741	5214	6203	4613	2333
0214	6254	5221	6204	4515	0000
0221	6571	5226	5226	0000	0000
0226	6573	5233	5233	0000	0000
0233	6131	5240	6135	4515	2666
0240	6041	5245	6042	4515	0306
0245	6031	5252	6036	4515	0313
0252	6431	5257	6432	4515	0000
0257	6421	5264	6426	4515	0000
0264	6471	5271	6472	4515	0000
0271	6461	5276	6466	4515	0000

CORMAP

0000	0100
0001	0077
0002	0001
0003	0002
0004	0003
0005	0004
0006	0005
0007	0006

TIME OUT QUEUE:

NODE	NEXT TASK VAL
------	---------------

FIELD TABLE:

0000	6707
0001	6224
0002	6230
0003	6247
0004	4250
0005	4260
0006	4270
0007	0000
0010	0000
0011	0000
0012	0000
0013	0000
0014	0000
0015	0000
0016	0000
0017	0000
0020	0000
0021	0000
0022	0000
0023	0000
0024	0000
0025	0000
0026	0000
0027	0000
0030	0000
0031	0000
0032	0000
0033	0000
0034	0000
0035	0000
0036	0000
0037	0000
0040	0000
0041	0000
0042	0000
0043	0000
0044	0000
0045	0000
0046	0000
0047	0000
0050	0000
0051	0000
0052	0000
0053	0000
0054	0000
0055	0000
0056	0000
0057	0000
0060	0000
0061	0000
0062	0000
0063	0000
0064	0000
0065	0000
0066	0000
0067	0000
0070	0000
0071	0000

0072	0000
0073	0000
0074	0000
0075	0000
0076	0000
0077	6210

AVAIL CHAINS.
FREE CORE CHAIN!
[0316 0000 0305 -0033

FREE NODES IN AVAIL CHAINS:

AVAIL2	0
AVAIL3	1
AVAIL5	82
AVAIL20	1

STATIC TASK LIST AND TASKS

STL[0]: GLOBAL.IDLE
5113 7777

TCB[6:11]: 0000 0000 0000 2352 0000 0000
WAITWORD: 0020 LINKWORD: 0000

STL[1]: GLOBAL.SWAP
5035 7777

TCB[6:11]: 0001 0000 0000 2042 0000 0000
MESSAGES IN REPORT QUEUE, WAITED RP:-2037
WAITWORD: 4000 LINKWORD: 5035

STL[2]: GLOBAL.FTCH
5051 7777

TCB[6:11]: 0002 0000 0000 2402 0000 0400
TCB[12:15]: 5167 5173 0000 0000
MESSAGES IN RECEIVE QUEUE, CLAIM WORD: 0000
MESSAGES IN REPORT QUEUE, WAITED RP:-
WAITWORD: 2002 LINKWORD: 5051

STL[3]: GLOBAL.TIME
5067 7777

TCB[6:11]: 0003 0000 2666 2600 0000 0000
MESSAGES IN REPORT QUEUE, WAITED RP:-2666
WAITWORD: 4000 LINKWORD: 5067

STL[4]: GLOBAL.SDSK
5103 7777

TCB[6:11]: 0004 0000 0000 2103 0000 0000
MESSAGES IN RECEIVE QUEUE, CLAIM WORD: 0000
MESSAGES IN REPORT QUEUE, WAITED RP:-
WAITWORD: 2000 LINKWORD: 5103

STL[5]: TEST .RTTY
5147 4002

TCB[6:11]: 0005 0303 0000 0604 0000 0000
TCB[12:15]: 0240 0201 0403 0024
MESSAGES IN RECEIVE QUEUE, CLAIM WORD: 5127
MESSAGES IN REPORT QUEUE, WAITED RP:-0313
WAITWORD: 4010 LINKWORD: 5147

STL[6]: GLOBAL.TSLD
0001 1437

STL[7]: GLOBAL.LDTS
0022 2001

STL[8]: TEST .WTTY
5167 4001

TCB[6:11]: 0010 0303 0000 1032 0000 0000
TCB[12:15]: 0000 1327 1003 0026
MESSAGES IN RECEIVE QUEUE, CLAIM WORD: 5147
MESSAGES IN REPORT QUEUE, WAITED RP:-
WAITWORD: 2010 LINKWORD: 5167
UNUSED ENTRIES IN STL: 55

SYSTEM STATISTICS.

INTERRUPTS: 5770
 SCHEDULES INTERRUPTING ACTIVE TASKS: 3
 SCHEDULES AT MONITOR EXIT: 2163
 MAX LENGTH OF RECEIVE/REPORT QUEUE: 81
 SD NORMAL SWAPS: 16
 SD VFIELD SWAPS: 0
 SWAP ERRORS: 0
 FIELDS USED: 4

MONITOR CALLS

STALL	0
SEND MS	587
WAIT REPORT	1798
SND MS AND WAIT	40
ERROR	0
REPORT MS	624
WAIT FOR MS	628
DISABL DEVINTR	0
SBR TO INTR	0
CLAIM INTR	2
FREE INTR	0
SIMULATE INTR	16
REQ PAGES	27
RTN PAGES	26
REQ FIELD	2
RTN FIELD	2
REQ ENTR IN STL	4
REQ TSK CONTRBL	0
RTN TSK CONTRBL	0
LOCK DF IN CORE	0
UNLOCK DF	0
REQ SPECIAL CDF	55
EXIT	4
STOP	0
RESUME	0
ERROR	0
FILL TCB	0
ERROR	0
ERROR	0
ERROR	0
ERROR	0
ERROR	0

NODES USED IN AVAIL CHAINS.

AVAIL2 0
 AVAIL3 2
 AVAIL5 84
 AVAIL20 3
 LOST SPACE: 3
 TOTAL AMOUNT OF SPACE USED OR LCST: 477

RUNNED DURING: 0 HRS 6 MIN 13.2 SEC
 IDLE TIME: 0 HRS 6 MIN 06.2 SEC
 IDLE DURING FIELD SWAP: 0 HRS 0 MIN 00.0 SEC

APPENDIX E.

LIST OF MONITOR COMMANDS.

CALLING SEQUENCE IN TASK:

```
CALL
    MONFUNC /COMMANDWORD
    ...    /POSSIBLE ARGUMENTS.
    ...    /NORMAL RETURN.
```

THERE ARE ABOUT 25 MONITOR COMMANDS DISTINGUISHED BY THE COMMANDWORD. THE COMMAND IS SPECIFIED IN BIT6-10 OF THE COMMANDWORD. BIT11 OF THE COMMANDWORD IS SET IF THE COMMAND USES ARGUMENTS.

SOME COMMANDS ACCEPT OPTIONS THAT ARE SPECIFIED IN THE OTHER BITS OF THE COMMANDWORD.

IN GENERAL COMMANDS DO PRESERVE LINK AND AC, UNLESS OTHERWISE STATED.

LIST OF MONITOR COMMANDS.

STALL

INSERT TASK IN TIMEOUTQUEUE (SEC. 1.9.6).
AC = MINUS TIMEOUT VALUE.
OPTIONS: SWPOUT,
AC := 0.

SENDMS

SEND A MESSAGE TO A TASK (SEC. 1.2).
ARG1 = MSPTR.
ARG2 = STATIC TASKNUMBER OF RECEIVERTASK.
OPTIONS: NONREP, DFPARM,

WTRP

WAIT FOR A REPORT (SEC. 1.2).
ARG1 = MSPTR OF WAITED REPORT.
OPTIONS: TIMEOUT, SWPOUT.
WAIT FOR THE REPORT ON THE SPECIFIED MESSAGE. IF NO MESSAGE WAS SPECIFIED (ARG1 = 0), WAIT FOR THE FIRST INCOMING REPORT.
AC := MSPTR OF REPORTED MESSAGE.
LINK := 1,

CHKRPQ

CHECK REPORTQUEUE (SEC. 1.2).

ARG1 =MSPTR OF WAITED REPORT.
 IF THE SPECIFIED REPORT IS PRESENT IN THE REPORTQUEUE, OR IN CASE
 ARG1 =0, IF ANY REPORT IS PRESENT IN THE REPORTQUEUE, AC :=MSPTR
 OF REPORT. ELSE AC:=0.
 LINK:=1.

SNDWTR
 SEND A MESSAGE AND WAIT FOR ITS REPORT (SEC. 1.2).
 ARG1 =MSPTR OF MESSAGE SENT.
 ARG2 =STATIC TASKNUMBER OF RECEIVERTASK.
 OPTIONS: DFPARM, TIMEOUT, SWPOUT.
 THE SPECIFIED MESSAGE IS SENT TO THE TASK AND THE SENDERTASK IS
 SET TO WAIT UNTIL THE MESSAGE IS REPORTED.
 AC:=MSPTR.
 LINK:=1.

RP
 REPORT A MESSAGE (SEC. 1.2).
 ARG1 =MSPTR OF REPORTED MESSAGE.
 IF THE NONREP OPTION WAS USED WHILE SENDING THIS MESSAGE, THE
 MESSAGE IS NOW RETURNED TO THE AVAILLIST SYSTEM, ELSE THE MESSAGE
 IS REPORTED TO THE SENDERTASK.

WTMS
 WAIT FOR A MESSAGE (SEC. 1.2).
 OPTIONS: KEEP, TIMEOUT, SWPOUT.
 ERASE PREVIOUS CLAIM, UNLESS KEEP OPTION WAS SPECIFIED, WAIT
 UNTIL A NEW MESSAGE ARRIVES.
 AC:=MSPTR OF RECEIVED MESSAGE.
 LINK:=0.

CHKRCQ
 CHECK RECEIVEQUEUE (SEC. 1.2).
 OPTION: KEEP.
 ERASE PREVIOUS CLAIM, UNLESS KEEP OPTION WAS SPECIFIED, IF A NEW
 MESSAGE HAS ARRIVED, AC:=MSPTR OF THAT MESSAGE. ELSE AC:=0.
 LINK:=0.

DISINTR
 DISCONNECT FROM INTERRUPT (SEC. 1.5.3).
 AC =SKIPOT.
 UPDATE THE SKIPCHAIN SUCH THAT THAT FLAGS OF THE SPECIFIED DEVICE
 ARE IGNORED (DEVICE DISABLED STATUS).
 AC:=0.

CNINTR

CONNECT SUBROUTINE TO INTSLOT (SEC. 1.5.3).

AC =SKIPLOT,

ARG1 =POINTER TO ENTRYPOINT OF SUBROUTINE,

ARG2 =MSPTR OF COMMUNICATION MESSAGE.

THE SUBROUTINE MUST RESIDE IN THE CURRENT DATAFIELD OF THE TASK,
IF A COMMUNICATION MESSAGE IS SPECIFIED (ARG2 'NE' 0), THIS
MESSAGE MAY BE REPORTED FROM THE CONNECTED ROUTINE TO THE TASK.

AC:=0.

CLINTR

CLAIM AN INTSLOT (SEC. 1.5.3).

AC =SKIPLOT,

ARG1 =CLEARLOT.

ARG2 =MSPTR OF MESSAGE.

ATTACH THE SPECIFIED MESSAGE TO THE INTSLOT OF THE SPECIFIED
DEVICE, THIS MESSAGE WILL BE REPORTED TO THE TASK EACH TIME THE
CORRESPONDING DEVICE INTERRUPTS.

AC:=0.

FRINTR

SET INTSLOT INTO DISCARD MODE (SEC. 1.5.3).

AC =SKIPLOT,

ARG1 =CLEARLOT.

WHEN THE INTSLOT IS IN DISCARD MODE, INTERRUPTS OF THE
CORRESPONDING DEVICE ARE DISCARDED.

AC:=0.

SIMINTR

SIMULATE INTERRUPT (SEC. 1.5.4).

ARG1 =ENTRYPOINT OF DEAF SECTION.

ENTER A DEAF SECTION AS IF IT WERE ENTERED FROM THE INTERRUPT
SECTION, IT MUST BE LEFT USING THE IEXIT INSTRUCTION, THE SECTION
MUST RESIDE IN THE CURRENT DATAFIELD OF THE TASK.

REQPAG

REQUEST A BUFFER OF 1-37 OCT CONSECUTIVE PAGES OF CORE
(SEC. 1.8.4).

AC =0.

ARG1 BIT5 =CORERESIDENT CONDITION.

ARG1 BIT7-11 =LENGTH OF REQUESTED BUFFER.

IF THE CORERESIDENT CONDITION IS SPECIFIED (BIT5 SET) THE BUFFER
WILL BE ALLOCATED IN A CORERESIDENT FIELD.

AC BIT0-4 :=NUMBER OF FIRST PAGE OF BUFFER.

AC BIT6-11 :=VIRTUAL FIELDNUMBER OF BUFFER.

RTNPAG

RETURN PAGES PREVIOUSLY REQUESTED USING THE REQ PAG COMMAND
(SEC. 1.8.4).

AC BIT0-4 =PAGENUMBER OF FIRST PAGE OF RETURNED BUFFER.

AC BIT6-11 =VIRTUAL FIELDNUMBER OF RETURNED BUFFER.

ARG1 BIT7-11 =NUMBER OF PAGES RETURNED.

AC:=0.

REQFLD

REQUEST A FIELD FOR DATA STORAGE (SEC. 1.8.2).

AC:=VIRTUAL FIELDNUMBER.

RTNFLD

RETURN A FIELD (SEC. 1.8.2).

AC =VIRTUAL FIELDNUMBER OF RETURNED FIELD.

AC:=0.

REQSTL

REQUEST ENTRY IN STL (SEC. 2.1).

ARG1, ARG2 =CONTENTS OF THE REQUESTED ENTRY.

SEARCH A FREE ENTRY IN THE STL AND DENOTE THE INDICATED CONTENTS
IN IT. RETURN THE CORRESPONDING STATIC TASKNUMBER IN AC.

AC:=STATIC TASKNUMBER.

REQTCB

ATTACH TCB TO ENTRY IN STL, OR RETRIEVE TCB ATTACHED TO ENTRY IN
STL (SEC. 2.1).

AC =STATIC TASKNUMBER.

IF A TCB WAS NOT YET ATTACHED TO THE TASK, THEN ATTACH ONE,
PREFILLED WITH INITIAL VALUES.

AC:=TCBPTR.

RTNTCB

DETACH TCB FROM TASK AND RETURN IT TO THE AVAILLIST (SEC. 2.1).

AC =STATIC TASKNUMBER.

AC:=0.

LOCK

LOCK DATAFIELD IN CORE (SEC. 1.4.4).

UNLOCK

UNLOCK DATAFIELD (SEC. 1.4.4).

REQCDF

REQUEST QUICK ACCESS TO DATAFIELD (SEC. 1.4.2).

AC =REQUESTED DATAFIELD.

SETS THE DATAFIELD LOADED INTO THE DFR AT THE EXECUTION OF A
VRCDF PSEUDO INSTRUCTION.
NOTE: AC REMAINS UNCHANGED!

EXIT
TERMINATE THE EXECUTION OF THIS TASK (SEC. 1.1.2).
TASK IS RESET INTO INITIAL STATUS. A NEW MESSAGE SENT TO IT WILL
START UP A FRESH COPY OF THE TASK.

STOP
STOP A TASK (SEC. 1.3.2).
AC =STATIC TASKNUMBER,
AC:=0.

RESUME
RESUME A TASK (SEC. 1.3.2).
AC =STATIC TASKNUMBER,
AC:=0.

FILTCB
WRITE VALUES INTO THE TCB OF A TASK (SEC. 2.1).
AC =STATIC TASKNUMBER,
ARG1 =POINTER TO VALUE LIST.
IF A TCB IS NOT YET ATTACHED TO THE TASK, THEN ATTACH ONE. FILL
THE FIRST 15 OCT (ALL BUT THE LAST) ENTRIES IN THE TCB WITH THE
VALUES INDICATED IN THE VALUE LIST. THE VALUE LIST MUST RESIDE IN
THE INSTRUCTIONFIELD OF THE TASK.
AC:=0.

LIST OF OPTIONS,

NONREP
NONREPORT (SEC. 1.2).
THIS OPTION MAY BE SPECIFIED WHEN A MESSAGE IS SENT. IF IT IS,
THE MESSAGE WILL BE RETURNED TO THE AVAILLIST SYSTEM WHEN IT IS
REPORTED. IF THE OPTION WAS NOT SPECIFIED, THE MESSAGE IS
REPORTED TO THE SENDER.

KEEP
KEEP PREVIOUS CLAIM (SEC. 1.2.1).
WHEN SPECIFIED IN CONJUNCTION WITH A WTMS OR CHKRCQ COMMAND, THE
TASK REMAINS CLAIMED BY THE SAME TASK AS BEFORE. OTHERWISE THIS
CLAIM TERMINATES.

SWPOUT

SWAP TASKCODE OUT (SEC. 1.9.4).
WHEN SPECIFIED IN CONJUNCTION WITH ONE OF THE COMMANDS WTMS,
WTRP, SNDWTR, OR STALL, THE PAGES OCCUPIED BY THE TASKCODE ARE
RETURNED IF THE TASK IS SET TO WAIT. A FRESH COPY OF THE TASKCODE
IS SWAPPED IN, AS SOON AS THE TASK IS RUNNABLE AGAIN.

TIMEOUT

RESTART TASK AFTER A WHILE (SEC. 1.9.6).
IF THIS OPTION IS SPECIFIED IN CONJUNCTION WITH THE COMMANDS
WTMS, WTRP, SNDWTR, THE TASK IS RESUMED AFTER THE SPECIFIED DELAY
EVEN IF THE OTHER WAITCONDITIONS ARE NOT YET FULFILLED, MINUS THE
TIMEOUT VALUE (LENGTH OF DELAY IN UNITS OF 0.1 SEC) MUST BE
SPECIFIED IN AC. IF ONE OF THE OTHER WAITCONDITIONS EXPIRES
BEFORE THE END OF THE DELAY, THE TIMEOUT OPTION HAD NO EFFECT,
OTHERWISE THE TASK IS RESUMED AFTER THE DELAY WITH AC=0.

DFPARM

ACTUAL DATAFIELD AS PARAMETER (SEC. 1.2, SEC. 1.9.2).
WHEN THIS OPTION IS SPECIFIED IN CONJUNCTION WITH ONE OF THE
COMMANDS SNDMS OR SNDWTR, THE ACTUAL DATAFIELDNUMBER IS COPIED
INTO BIT6-8 OF MESSAGE[2] (FIRST INFORMATIONWORD).

THE TEXT PRESENTED BELOW MAY BE USED TO BUILD THE MC8 SYSTEM
UNDER CONTROL OF THE OS/8 OPERATING SYSTEM, USING
THE PROGRAM BATCH.SV.
THE PROGRAM GLOBL.SV SELECTS SYMBOLS FROM THE PAL8 SYMBOLTABLE
AND ARRANGES THEM IN A 'DIRECT ASSIGNMENT' FILE.
THIS FILE IS USED AS PREFILE WHEN ASSEMBLING TASKS.
OP.PA IS A PREFILE HOLDING OBVIOUS DEFENITIONS.

```
$JOB BUILD MC8 FOR PDP8E.  
.PAL MC,MC<OP,MCCONF.E,MC81,MC82,MC83,MC84,MC85/N/D/H  
.R GLOBL  
*MC8.SY<MCTSYM,MC.LS  
.R PIP  
*OPMC8E,PA<OP.PA,MC8.SY$  
.R GLOBL  
*MC8SYS.SY<MCSYSM,MC.LS  
.PAL MC8HLT<OPMC8E,PA,MC8HLT  
.PAL MC8IN<OP,MC8SYS.SY,MC8IN  
.R ABSLDR  
*MC,MC8HLT,MC8IN$  
.SA SYS MC8  
.SA SYS FLD0 0-5000  
.PAL OP,MCCONF.E,MC8SYS.SY,MC8BUL,IO/L=10200  
.SA SYS MC8BUL  
.PAL OP,MCCONF.E,MCTBLD,IO/L  
.SA SYS MCTBLD  
.R MC8PAL  
*OP,MCCONF.E,MC8SYS.SY,MCTSLD/L/Z$  
.SA SYS MCILIB 20000-27777;25000  
.R MCILIB  
.R MC8BUL  
*X.TM<SYS:FLD0/Y/Z  
$END
```

FEB 21 11:08 1977 TEST PAGE 1

```
*****
*           *
*   *****   *
*   *           *
*   *           *
*   *           *
*   *           *
*   *           *
*   *           *
*   *           *
```

```
PROGRAM TEST(INPUT,OUTPUT);
BEGIN
WRITELN('SOME ', 'CONSTANTS');
WRITELN();
WRITELN(1.0E50,1.0E+50,-1.0E50,-1.0E+50);
WRITELN(1.0E38,1.0E+38,-1.0E38,-1.0E+38);
WRITELN(1.0E10,1.0E+10,-1.0E10,-1.0E+10);
END.
```

PC TEST

```
SCAN 1: 115 (=0163) BYTES FOR CODE
SCAN 2: 113 (=0161) BYTES FOR CODE
SCAN 3: 113 (=0161) BYTES FOR CODE
30 (=036) BYTES FOR CONSTANTS
```

PASCAL

SOME CONSTANTS

```
+1.694765E+38 *1.000000E+00 -1.694765E+38 -1.000000E+00
+9.999999E+37 *1.000000E+00 -9.999999E+37 -1.000000E+00
+1.000000E+10 *1.000000E+00 -1.000000E+10 -1.000000E+00
```

FEB 21 11:16 1977 TST PAGE 1

```
*****  *****  *****
*      *      *
*      *      *
*      *      *
*      *      *
*      *      *
*      *      *
```

```
PROGRAM TEST (INPUT,OUTPUT);
BEGIN
WRITELN('OUTPUT!');
WRITELN(1.0E10,1.0E+10,-1.0E10,-1.0E+10)
END.
OUTPUT:
+1.000000E+10 +1.000000E+00 -1.000000E+10 -1.000000E+00
```

ONTVANGEN 17 MAART 1977