1. TOSS MONITOR

1.1. Introduction

This chapter describes briefly the structure of the TOSS monitor and the relationships between the Monitor components.
The reasons for including such a description in this Manual are:

- To enable the application programmer to understand the effects of Monitor requests.
- To illustrate the relationships between tasks, Monitor and hardware. This information is especially useful to programmers who wish to generate a TOSS Monitor.
- To introduce TOSS specialist programmers to the general concepts of the TOSS Monitor.

1.2. General Description

The TOSS Monitor is divided into the following logical functions:

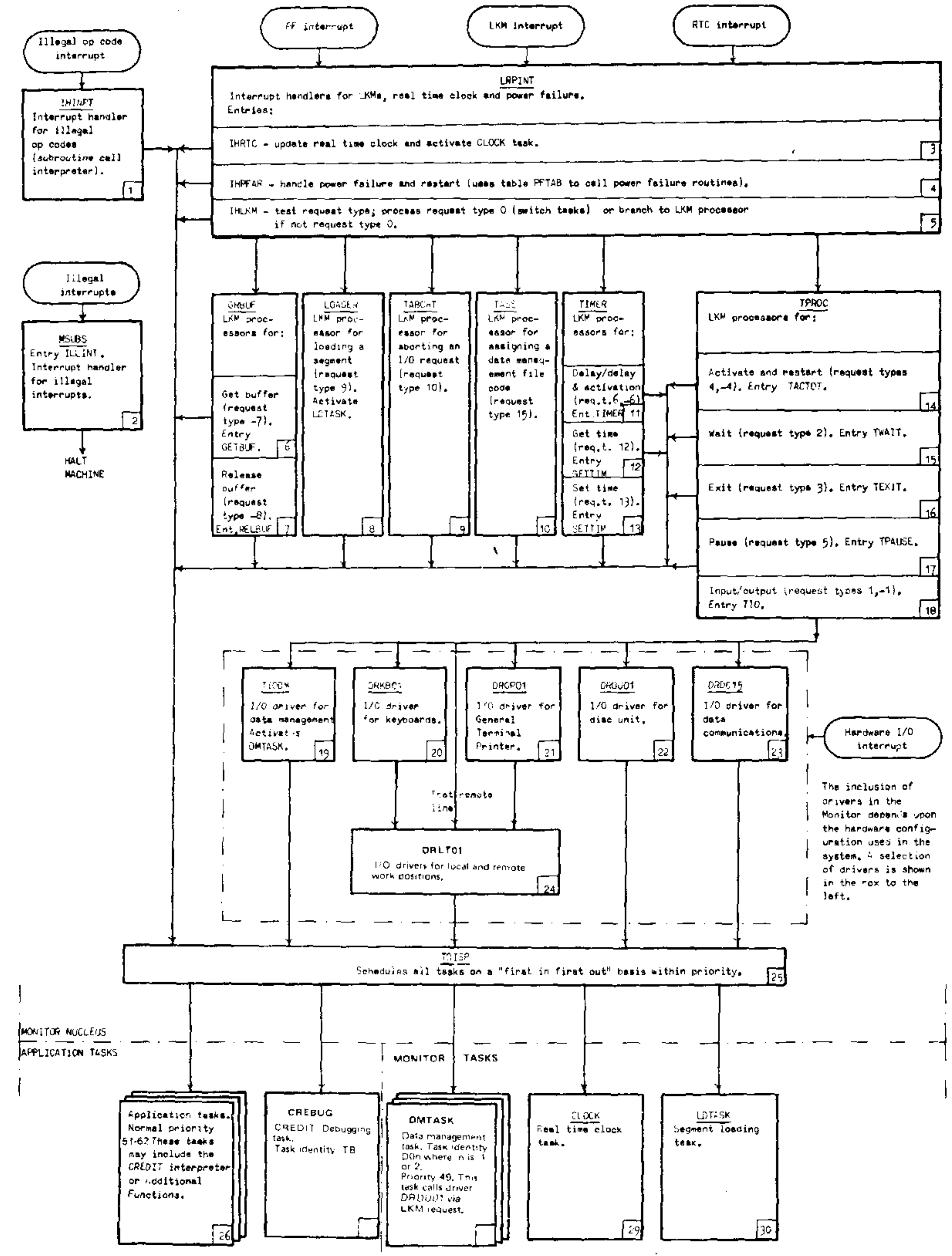
- Interrupt handlers
- LKM processors
- I/O drivers
- Dispatcher
- Tables
- Other Monitor programs.

Reference will be made to the diagram on the next page which illustrates the relationship between the various components of the Monitor. Note that this diagram does not show every module in the Monitor. The diagram has been simplified so that only the major modules and entry points are shown.

The following components are not considered:

- Monitor Initialisation Program
- Monitor Configuration Program
- Assembler Debugging Program

Reference numbers appear in the corner of each box in the diagram. These numbers are referred to in the following sections.
These boxes are grouped into modules. Each box represents an entry point in the containing module. The module name is shown in the top box of the module.



Illegal op code interrupt
IHINPT Interrupt handler for illegal op codes (subroutine call interpreter). 1
FF interrupt
LKM interrupt
RTC interrupt
LRPINT Interrupt handlers for LKMs, real time clock and power failure. Entries:
IHRTC - update real time clock and activate CLOCK task. 3
IHPFAR - handle power failure and restart (uses table PFTAB to call power failure routines). 4
IHLKM - test request type; process request type 0 (switch tasks) or branch to LKM processor if not request type 0. 5
Illegal interrupts
MSUBS Entry ILLINT. Interrupt handler for illegal interrupts. 2
HALT MACHINE
GRBUF LKM processors for:
Get buffer (request type -7). Entry GETBUF. 6
Release buffer (request type -8). Ent. RELBUF 7
LOADER LKM processor for loading a segment (request type 9). Activate LDTASK. 8
TABORT LKM processor for aborting an I/O request (request type 10). 9
TASS LKM processor for assigning a data management file code (request type 15). 10
TIMER LKM processors for:
Delay/delay & activation (req.t.6,-6) Ent.TIMER 11
Get time (req.t. 12). Entry GETTIM 12
Set time (req.t. 13). Entry SETTIM 13
TPROC LKM processors for:
Activate and restart (request types 4,-4). Entry TACTOT. 14
Wait (request type 2). Entry TWAIT. 15
Exit (request type 3). Entry TEXIT. 16
Pause (request type 5). Entry TPAUSE. 17
Input/output (request types 1,-1). Entry TIO. 18
TIODM I/O driver for data management Activates DMTASK. 19
DRKB01 I/O driver for keyboards. 20
DRGP01 I/O driver for General Terminal Printer. 21
DRDU01 I/O driver for disc unit. 22
DRDC15 I/O driver for data communications. 23
Hardware I/O interrupt
The inclusion of drivers in the Monitor depends upon the hardware configuration uses in the system. A selection of drivers is shown in the box to the left.
That remote line
DRLT01 I/O drivers for local and remote work positions. 24
TDISP Schedules all tasks on a "first in first out" basis within priority. 25
MONITOR NUCLEUS
APPLICATION TASKS
MONITOR TASKS
Application tasks. Normal priority 51-62 These tasks may include the CREDIT interpreter or additional Functions. 26
CREBUG CREDIT Debugging task. Task identity TB
DMTASK Data management task. Task identity D0n where n is 1 or 2. Priority 49. This task calls driver DRDU01 via LKM request.
CLOCK Real time clock task. 29
LDTASK Segment loading task. 30

1.3. Interrupt Handlers

An "interrupt" is an event which causes control to be passed, at the completion of the current instruction, to one of the "interrupt vectors" held in memory words 0 to 63.
An interrupt is automatically generated in response to one of the following events:

- Power failure
- LKM request
- Real time clock update
- Execution of an illegal instruction
- Completion of I/O action

The particular vector entry used depends upon the type of interrupt.
Each vector entry contains a pointer to an associated "interrupt handler".

For example memory word 0 contains the interrupt vector for power failure. When power failure occurs the sequence of instructions currently being executed is interrupted, and control is handed to the instruction pointed to by memory word 0 (in the power failure interrupt handler).

As can be seen from the above diagram the interrupt handlers in boxes 1, 2 and 4 are self contained : they do not call any subsidiary modules. The handler in box 2 terminates by halting the machine. The handlers in boxes 1 and 4 terminate by branching to the Dispatcher (box 25).

The interrupt handler for the real time clock (box 3) activates a special clock task (box 29) every 100 milliseconds.
The handler terminates by branching to the Dispatcher (box 25). The interrupts for completion of I/O action are serviced by the appropriate device driver (box 22 for example).

The interrupt handler for LKM requests processes only request type 0. The interrupt handler saves registers A1 to A14 and if necessary branches to one of several "LKM processors" to process the remaining types of LKM requests. These processors are described in the following section.

1.4 LKM Processors

The LKM processors perform the following functions:

- Task scheduling (boxes 11, 14, 16 and 17)
- Input/output (boxes 9, 10, 15 and 18)
- Buffer control (boxes 6 and 7)
- Memory management (box 8)
- Monitor clock control (boxes 12 and 13)

A particular processor is invoked as a result of a LKM instruction executed in a task. As described in section 1.3, this instruction causes an interrupt. The DATA directive following the LKM instruction contains a numeric request type.
This is used by the interrupt handler (box 5) to select the required LKM processor (boxes 6 to 18).

The LKM processors in boxes 6, 7, 9, 11, 12 and 13 (i.e. modules GRBUF, TABORT and TIMER) are only included in the Monitor if they are specifically requested during system generation. The LKM processor in box 8 (i.e. module LOADER) is only included if memory management is requested during system generation.
The LKM processor in box 10 (i.e. module TASS) is only included if data management is requested during system generation. The LKM processors in boxes 6, 7 and 9 to 17 are self contained. They do not call any subsidiary modules. These processors terminate by branching to the Dispatcher (box 25).

The LKM processor for segment loading (box 8) activates a special loading task (box 30), via a normal LKM request, to carry out the loading process. This processor terminates by branching to the Dispatcher (box 25).

The LKM processor for I/O control (box 18) branches to the appropriate I/O driver (boxes 19 to 24 for example) to execute the I/O request. I/O drivers are discussed in the next section.

1.5 I/O drivers

I/O drivers perform device control functions and (optionally) process data communications and data management I/O requests. The type of function to be performed is specified in register A7 by the requesting task. Examples are as follows:

/00	Test Status
/01	Basic Read
/02	Standard Read
/03	Numeric Read
/05	Basic Write
/06	Standard Write
/0A	Random Read
/0B	Random Write

Further parameters are specified in an "event control block".
The address of the event control block is placed in register A8 by the current task.
A separate driver is available for each type of device (for example boxes 20 to 22 in the diagram) and some devices may have more than one driver (for example keyboard). The required drivers must be built into the Monitor during system generation. Separate drivers are also available for performing device control (e.g. box 24) at either local or local and remote work positions comprising one or more of the following devices:

- Keyboard
- Teller Terminal Printer
- General Printer
- Numeric Display
- Indicator Display/Keyboard Lamps
- Video/Plasma Display

The devices must be connected to the Terminal Computer via a Channel Unit for Local Terminals (CHLT) and/or a Channel Unit for Remote Terminals (CHRT). Devices which are used locally (i.e. not via modems) must be connected to the CHLT. Devices which are used remotely (i.e. via modems) must be connected to the CHRT.

One of two drivers may be used to control devices attached to the CHLT and CHRT. Driver DRLT01 is used to control devices attached to the CHLT. Driver DRRT01 is used to control devices attached either to the CHLT or CHRT. That is driver DRLT01 controls locally connected devices only, and driver DRRT01 controls both locally and remotely connected devices. Only one of these drivers are included in the Monitor during system generation.

Driver DRLT01 or DRRT01 is normally entered only from the individual device drivers for the above devices (for example boxes 20 and 21). The only exception to this rule is the "test remote line" function. In this case the driver DRRT01 is entered directly from the LKM processor (box 18).

Drivers DRLT01 and DRRT01 terminate by branching to the Dispatcher (box 25).

The drivers for the remaining devices not listed above (for example box 27) are self contained. That is they do not enter any additional driver. These drivers all terminate by branching to the Dispatcher.

A separate driver is also available for each type of communication line discipline. Only one type of communication driver can be included in the Monitor at one time (for example box 23). These drivers terminate by branching to the Dispatcher (box 25).

I/O drivers for devices and data communication are only included in the Monitor if they are specifically requested during system generation.

A separate driver TIODM is available for data management (box 19). This driver activates a special data management task (box 27 – one task per disk drive) to process data management requests. This task issues a disk I/O LKM request to perform the actual I/O via the disk driver (box 22). Driver TIODM terminates by branching to the Dispatcher (box 25). The data management driver is only included in the Monitor if data management is requested during system generation.

1.6 The Dispatcher

Previous sections have described the way in which interrupts (LKM request, completion of I/O action etc.) are processed by the Monitor. When the processing of an interrupt is complete, control is handed to the Dispatcher.

The Dispatcher then determines which application or Monitor tasks are able to proceed. If several tasks are able to proceed a task is chosen on a "first in first out" basis within priority level. Registers A1 to A14 are restored for this task and the task is entered via a RTN instruction.

The RTN instruction automatically enables interrupts to occur. Any interrupt of an equal or lower priority which occurred during the processing of the last interrupt would have been queued. On the RTN instruction being executed this interrupt will take effect immediately and control will again be passed to an interrupt handler.

However, if no interrupt has been queued the task will begin execution. Execution of the task will continue until another interrupt occurs (which may of course be a LKM instruction executed by the task). When processing of this interrupt has been completed the task will again become a candidate for scheduling. And so on.

A task may exit by issuing a LKM request type 3. The Dispatcher will then delete all record of the task and the task will cease to exist.

Another task may be activated by the running task issuing LKM request type — 4.

A task may pause itself by issuing a LKM request type 5.

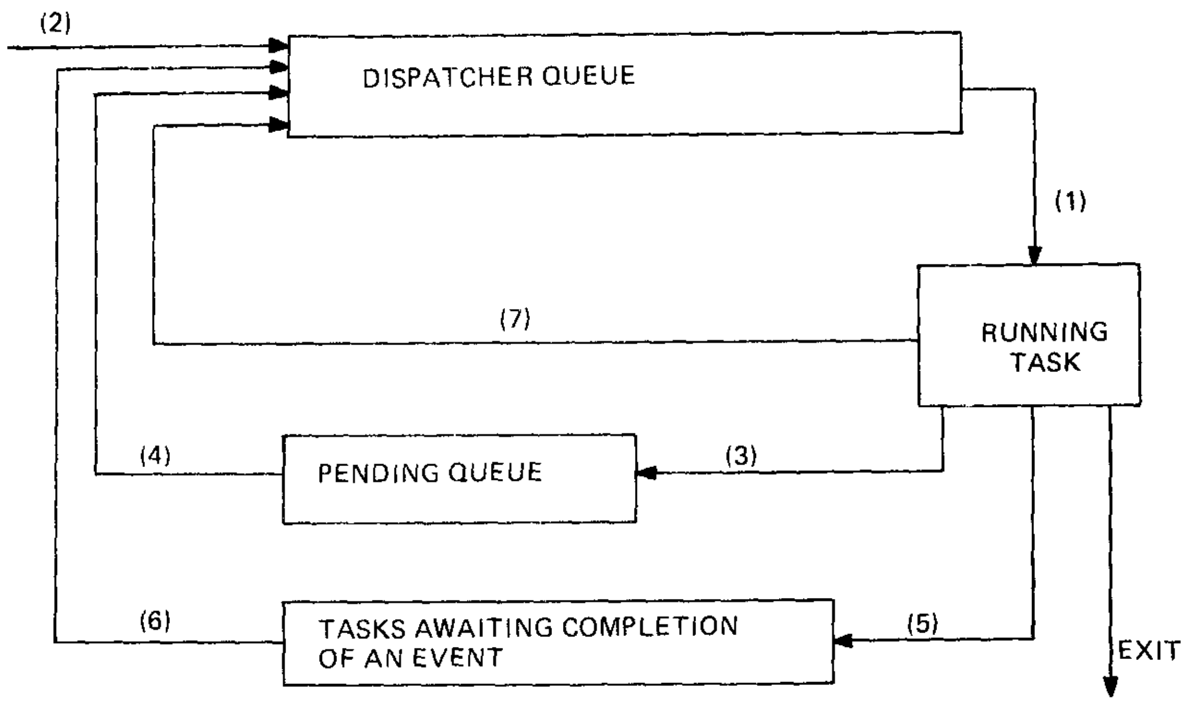
A paused task may then be restarted by another task issuing a LKM request type 4.

A task may be placed at the back of the dispatcher queue and cause the task at the head of the queue to be called into execution by issuing a LKM request type 0 (switch tasks).

Exit, activation, pause, restart and switch are all controlled by the LKM processors and Dispatcher.

In order to identify and schedule tasks, each task must have a task identifier and priority level. Application tasks must be assigned a task identifier and priority level during system generation. Monitor tasks (e.g. data management task, segment loading task) have a predefined task identifier and priority level and need not be specified during system generation.

The scheduling of tasks by the LKM processors and Dispatcher is illustrated in the following diagram.



The following notes refer to the numbers in the diagram:

1. Tasks are dispatched from the dispatcher queue "first in first out" within priority. That is, each task as it becomes available for dispatching is placed at the back of the dispatcher queue. When selecting a task for dispatching, the Dispatcher considers only the tasks with the highest priority level. The task at the front of the queue in this priority level is chosen for dispatching.
2. A running task may activate or restart another task. The new task is placed at the back of the dispatcher queue and the running task continues.
3. A running task may activate itself (normally at a different start point from the original one). However, the Dispatcher cannot schedule two tasks with the same task identity at the same time. For this reason the newly activated task is placed in a pending queue and the running task continues.
4. When the running task performs an exit LKM request the pending task with the same task identity is placed at the back of the dispatcher queue.
5. When the running task issues certain LKM requests (e.g. I/O) it must wait until the requested event is complete. During this time the Dispatcher will select another task from the dispatcher queue and the original task will not be considered for dispatching.

6. When the event is complete the waiting task is placed at the back of the dispatcher queue.
7. A running task may request that it be placed at the back of the dispatcher queue and that the task at the head of the queue be called into execution.

For an example of scheduling using the LKM request I/O and activation see appendix A.

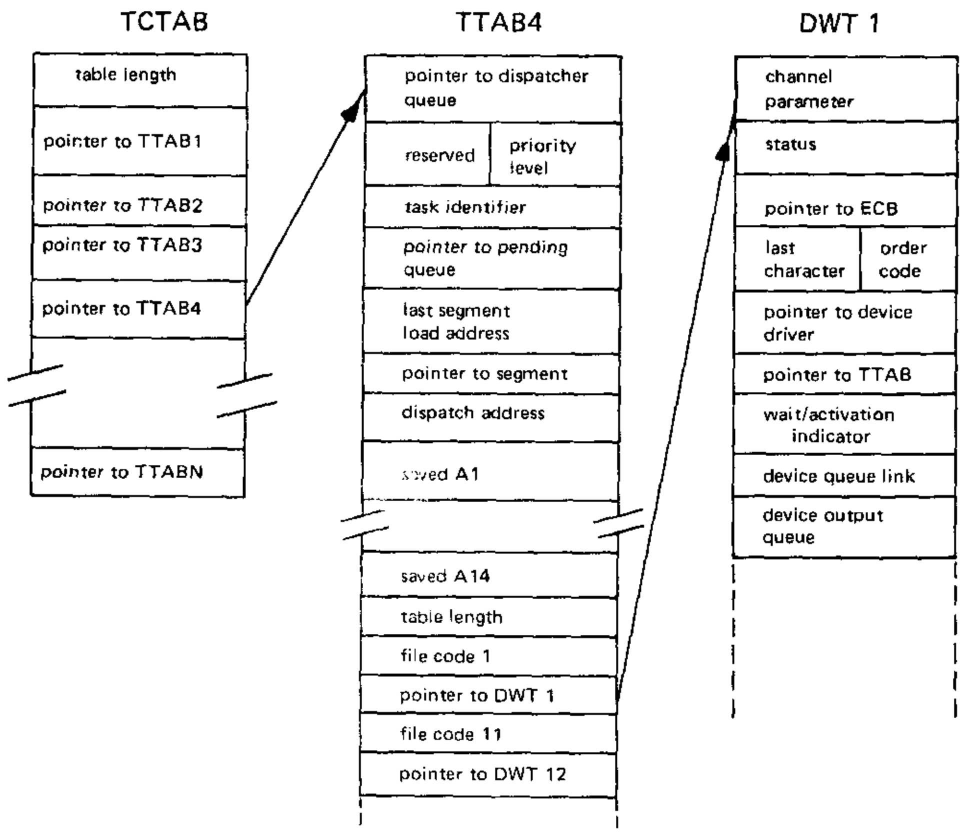
When the dispatcher queue is empty the Dispatcher simply performs an "idle loop" (priority level 63) waiting for a task to be queued. This situation will arise when, for example, all tasks are waiting for I/O to be completed. As soon as an I/O is completed a hardware interrupt will pass control to the appropriate device driver. The driver will then call a routine to queue the waiting task in the dispatcher queue and hand control to the Dispatcher.

1.7 Tables

The Monitor uses a large number of tables to record the status of the various components of the system.
Three of the more important tables are:

- Task control table (TCTAB)
- Task table (TTAB)
- Device work table (DWT)

The following diagram indicates briefly the contents of these tables and the relationships between them:



There is one TCTAB in the system. It points to each TTAB in the system. There is one TTAB per task. TTAB points to each DWT used by the task. There is one DWT for each device in the system, though the CREDIT debugger requires a second DWT for the SOP switches.
TCTAB, TAB's and DWT's are built up from the task definition information supplied to SYSGEN or MONCON. The pointers in TCTAB occur in the order in which tasks were defined during task definition.
The first task in TCTAB is the task activated by the Monitor when the application is first loaded into memory.

1.8. Other Monitor Programs

1.8.1 *Introduction*

The following additional programs (not shown on the above diagram) may be included in the TOSS Monitor if they are requested during system generation:

- Monitor Configuration Program
- Monitor Initialisation Program

These programs are discussed in the following sections.

1.8.2 *Monitor Configuration Program*

The Monitor Configuration Program (MONCON) is executed by the TOSS Monitor during the system start process. MONCON will read certain parameters from a Monitor configuration file supplied by the user. These parameters are used by MONCON to generate certain Monitor tables required for the application tasks.
MONCON is then overwritten in memory.

MONCON is only executed if it has been requested during system generation.
If MONCON is not requested the parameters must be keyed-in to SYSGEN during "task definition" and "common device definition".

The objective of MONCON is to provide the user with a quick and simple method of supplying certain parameters to the Monitor which are likely to change relatively frequently. These parameters may thus be altered without having to re-generate a TOSS Monitor.

1.8.3 *Monitor Initialisation Program*

This program enables the user to restart the TOSS Monitor at address /92 without having to reload the Monitor.

This facility is normally only used for program testing via the Assembler Debugging Program. It is only included in the Monitor if requested during system generation.

1.9. TOSS System software

In addition to the TOSS Monitor, other software components are available which run under the control of the Monitor. They are:

- CREDIT Configurator
- CREDIT Debugging Program
- CREDIT Interpreter
- TOSS Utility Programs
- Additional functions

These components, together with the TOSS Monitor, are known as TOSS System Software.

The CREDIT Configurator, Debugging Program and Interpreter are described in the CREDIT PRM (M04).

The TOSS utility programs are described in the TOSS User Specification (M17) and in chapter 4 of the present Manual.

The additional functions are described in the Assembler PRM (M06 Part 1).