A MULTI-USER PROCESS INTERFACE SYSTEM

FOR A

PROCESS CONTROL COMPUTER

by BARRY GRAHAM SHERLOCK

Submitted to the University of Cape Town in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

September 1983

ACKNOWLEDGEMENTS

My sincere appreciation is extended to:

Professor H.S. Bradlow, for his guidance, encouragement and patience while supervising this project.

Bill Randall, who supplied the air-column apparatus used in the student project, and was always ready to assist in all matters relating to the Chemical Engineering department.

Tony Eva, for introducing me to the use of the PDP-11, and for modification of the plotting terminal line from current loop to RS-232.

Alan Day, for valuable technical assistance including construction of the Media front panel.

Bruce Ingram of SANS for advice and documentation on using the Media hardware.

My brother Derek who assisted with the printing and copying of this thesis.

The CSIR, for their financial assistance during 1982.

Any others whom I may unwittingly have failed to mention here.

ABSTRACT

This thesis describes a system to implement a distributed multi-user process interface to allow the PDP-11/23 computer in the Electrical Engineering department at UCT to be used for process control. The use of this system is to be shared between postgraduate students for research and undergraduates for doing real-time control projects. The interface may be used concurrently by several users, and access is controlled in such a way as to prevent users' programs from interfering with one another.

The process interface hardware used was a GEC Micro-Media system, which is a stand-alone process interface system communicating with a host (the PDP-11/23) via a serial line. Hardware to drive a 600 metre serial link at 9600 baud between the PDP-11/23 and the Media interface was designed and built.

The software system on the host, written in RTL/2, holds all data from the interface in a resident common data-base and continually updates it. Access to the interface by applications programs is done indirectly by reading and writing to the data-base, for which purpose a library of user interface routines is provided.

To allow future expansion and modification of the Media interface, software (also written in RTL/2) for an LSI-11 minicomputer interfaced to the Media bus was developed which emulates the operation of the GEC proprietary Micro-Media software. A program to download this software into the LSI-11 was written.

A suite of diagnostic programs enable testing of the system hardware and software at various levels.

To ease testing, teaching and applications programming, a

general-purpose simulation package for the simulation of analogue systems was developed, as well as graphics routines for use with a Tektronix 4010 plotting terminal.

A real-time computing project for a class of undergraduates was run in 1983. This project made extensive use of the system and demonstrated its viability.

Keywords:

distributed	interface	multi-user
computer	process control	real-time

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Requirements

This project arose out of the need of the University of Cape Town's Electrical and Chemical Engineering departments for an interface system to enable them to use the PDP-11/23 minicomputer that is shared between the two departments, for process control. The interface system had to satisfy the following requirements:

1. Since the PDP-11 is used for various purposes, such as departmental administration, teaching and research, the use of the process interface must not load the computer unduly; i.e. its use must coexist with the other tasks performed by the computer.

2. The interface system should be suitable for use both for postgraduate research and for undergraduate teaching.

3. Multi-user access to the system should be possible, so that it can be used concurrently for postgraduate

research and undergraduate teaching. Typically, at any given time, the interface may be in use by one postgraduate and three or four out of a class of about fifty final-year undergraduates.

4. The system should be commissioned as quickly as possible, with as little effort as possible from University staff, since the University is badly short-staffed in this area.

5. Software access to the interface system must be carefully controlled, so as to ensure that no two users can accidentally or deliberately interfere with each other's software. In particular, the use of the system by undergraduate students must not in any way interfere with the research use by postgraduates.

6. Geographic considerations, viz. the fact that the PDP-11/23 is situated in the Electrical Engineering building and the processes to which it must interface are in the Chemical Engineering building 600 metres away, dictate that the sytem must be distributed.

1.2 The Process Interface System

1.2.1 Commercial System

A Media system [1] was chosen as the process interface hardware (see chapter 2 for a description of Media). This choice was based on the following considerations:

1. Media is available in various sizes, ranging from very small systems, such as the one used in this project, to very large industrial systems involving several thousands of inputs and outputs. From the teaching point of view, therefore, the students would be using a system which, although small and relatively inexpensive, is similar in architecture to large systems used in the industrial environment.

2. Media is essentially a modular system, since it consists of sets of card-cages or "bins" into which the various Media circuit cards (such as analogue or digital inputs and outputs) are plugged. This means that Media can be easily expanded by the addition of further Media bins or by replacing cards or inserting additional cards within a bin.

3. Media is widely used in industrial process control in the Western Cape, by such companies as AECI and SA Nylon Spinners (SANS). Since contact is maintained between the University and (particularly) SANS, the advantage would be gained of a productive interchange of knowledge between the University and industry.

4. Media provides fast access to data anywhere in the system, since the Media cards are mapped to word addresses on the Media bus or "highway".

General Electric Corporation (GEC) supply a small system, called the GEC Micro-Media system [2], which is a stand-alone Media process interface system which communicates with the outside world via a serial line, using a simple link-level protocol. The system may therefore be used as a remote stand-alone "outstation" controlled, via the serial line, by a host computer (in this case a PDP-11/23).

Stand-alone operation in Micro-Media is achieved by including three special Media cards: the 8085 CPU card, Media interface card and serial line interface card. The Media interface card causes the Media cards to be addressable by the 8085 on the CPU card, and the serial line interface card enables the 8085 to communicate via the serial line to the host. The 8085 CPU card has two ROMs containing a program which continually monitors the serial line, receives command frames from the host, decodes them, acts on them (for example, read a value from a particular analogue input), and encodes and sends the reply frames. Use of the GEC Micro-Media system with this built-in program enables the system to be commissioned very quickly because it is ready to run and only the host software to drive the serial link need be written.

Because of the simplicity of the serial link protocol used by the GEC Micro-Media system it is possible to write this host software without modifying the operating system device drivers, i.e. a normal serial port driver can be used provided that it supports a transparent mode which passes all characters (the ports of the PDP-11/23 can be configured to do this).

It was therefore decided to implement a GEC Micro-Media system. The configuration chosen was one with two Media bins containing the following Media cards:

1. A 10-bit analogue-to-digital converter card plus a 16-way multiplexer card, thus providing 16 multiplexed analogue inputs.

2. Two 16-line opto-isolated digital input cards.

3. A 16-line digital output card.

4. A 4-channel 8-bit digital-to-analogue converter card, thus providing 4 analogue outputs.

5. The 8085 CPU, Media interface and serial line interface

cards.

6. LSI-11-to-Media interface cards.

These cards provide sufficiently many inputs and outputs to support the applications envisaged for the system.

1.2.2 The Modified System

In addition to being interfaced to the 8085 as explained above, the Media highway may be interfaced to the LSI-11 Q-bus by means of the LSI-11-to-Media interface cards mentioned above. The use of these cards to interface an LSI-11 microcomputer to the Media highway enables the LSI to access all of the Media cards at memory speeds. This interface can coexist with the 8085 subsystem, i.e. it is not necessary to remove the 8085 CPU, Media interface or line interface cards if the LSI-11 interface is to be used.

A major aspect of this thesis is the development of an alternative software system [3] to enable an LSI-11 microcomputer to control Media as an alternative to the GEC 8085-based system. This system, which will henceforth be referred to as the "LSI-11 Media system", is software compatible from the host point of view with the GEC Media system and was developed because of the following considerations:

1. The documentation available on the GEC Media system was not sufficient to enable it to be easily modified and extended. The LSI-11 Media system, on the other hand, having been developed as part of this project, is fully documented and understood and can be modified to allow for future developments, such as the addition of further Media cards.

2. Although the GEC Media system is in theory immediately usable, there were in fact several practical difficulties in getting it to operate correctly. Difficulties encountered included a hardware problem whereby the system would re-initialise itself unexpectedly every few minutes thus destroying data held in RAM, and a software bug which made the analogue input data inaccessible.

As an alternative to developing an LSI-11 system which emulates the GEC Media system, we could have designed and developed an LSI-11 system without ensuring compatibility with the GEC system. This approach has the advantages that it could be more closely tailored to the needs of the application in mind, and that one would not be forced to live with the several deficiencies of the link-level protocol used by the GEC system (see chapter 2). However, it was felt that these disadvantages were outweighed by the fact that compatibility would enable the GEC Media system to be used as a back-up system in the event of a hardware failure in the LSI-11 Media system.

The LSI-11 Media system software emulating the operation of the GEC system was written in the high-level structured real-time language RTL/2 in the form of two tasks running under the operating system SMT [4]. This combination was chosen because:

1. The software could be developed on the host PDP-11/23 computer using its editing, RTL/2 compiling and linking utilities.

2. RTL/2 as a real-time language offers several attractive features including efficient code generation, legibility and ease of writing, debugging and modification.

3. SMT is a small (6K words) multitasking operating system ideally suited to downloading and running multi-task real-time applications on a small LSI-11 micromputer such

as the one used. It provides support for up to sixteen independent tasks written in RTL/2.

The LSI-11 Media system software, named SMT.TSK (after the operating system used), must be downloaded from the host computer using a program called ODT (see chapter 3). This program enables the operator to load a bootstrap loader into the LSI, and then, by starting the bootstrap loader running, download SMT.TSK and start it running. The necessity to download the program before use, although simple and quick, is a disadvantage of the LSI-11 Media system, since the GEC system program resides in ROM and does not need to be downloaded. This is however offset by the flexibility of being able to download specialised software systems if so desired.

1.3 Serial Data Link Hardware

Because of the relatively large distance of 600 metres between the Media interface and the host, an ordinary RS-232 link could not be used. The link is a 4-wire link installed by the GPO and is driven in differential mode using the 8820 and 8830 differential line driver and receiver integrated circuits. Two of these 4-wire lines are used: one for the Media interface and one for a remote terminal to the PDP-11. The line-driver hardware is described in chapter 4. The system hardware is illustrated in figure 1-1.

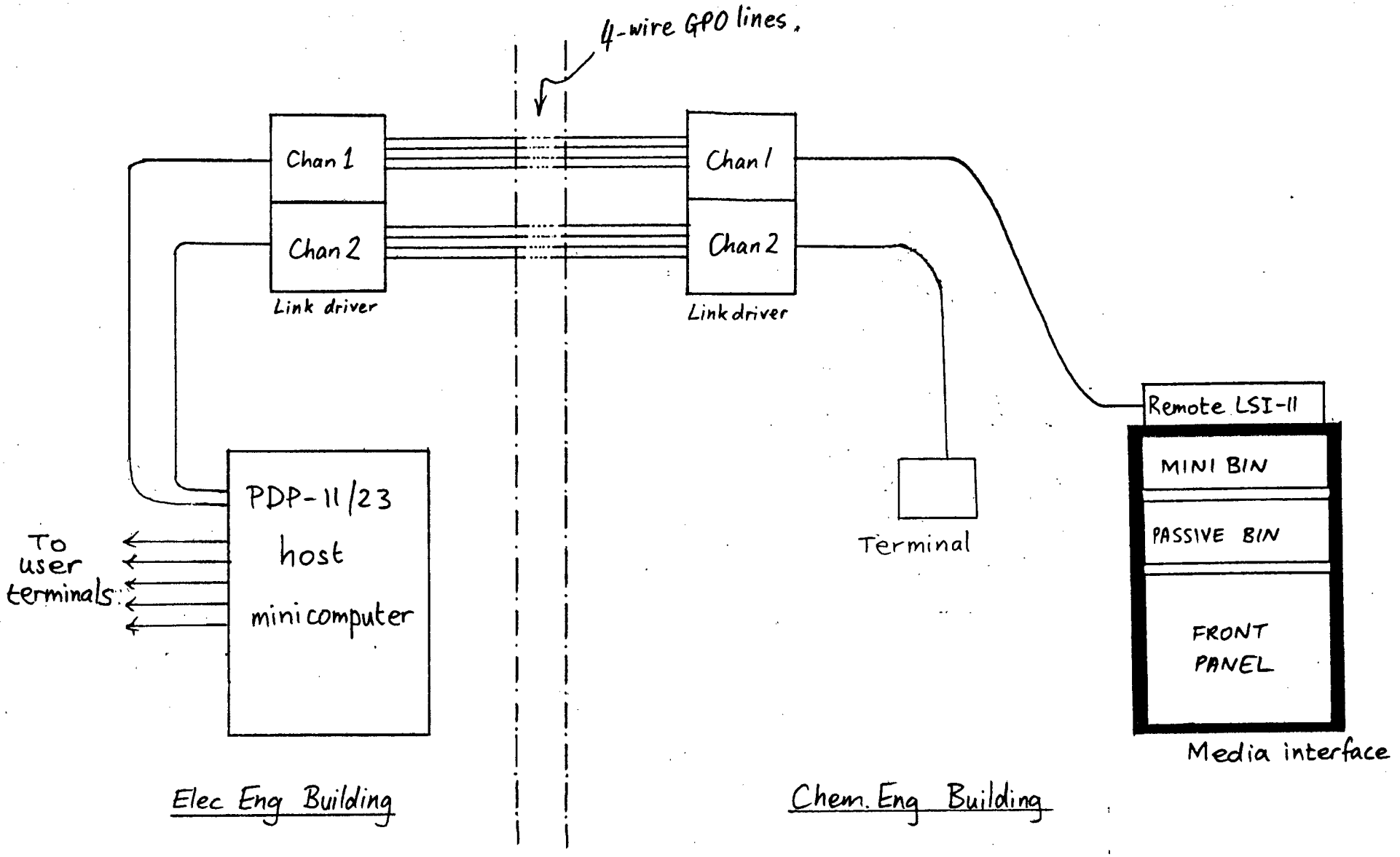


Fig 1-1: System hardware.

1.4 The Host Software System

1.4.1 Introduction

The PDP-11/23 host minicomputer runs the multi-user multi-tasking operating system RSX-11M [5] which provides the environment for the development and execution of multiple real-time tasks using a priority-structured, event driven task scheduler. Password log-in protection, memory protection and a heirarchical file structure make it suitable for use by many users (such as a class of students).

The host software system provides the necessary routines, libraries, diagnostics and tasks to enable user-friendly multi-user access to the Media interface. The host software, like the LSI-11 Media system software, is written in the language RTL/2 [6] , a structured high-level language for real-time systems. RTL/2 offers the advantages of compiling into efficient code, it has comprehensive error handling, it is easy to learn and can be used for both systems and applications programming. It was therefore felt that RTL/2 was a good choice both for the host software system and for the applications programs to be run using the system. Its use in industry (e.g. at AECI and SANS) means that students using RTL/2 will be gaining useful programming experience.

The RTL/2 compiler for the PDP-11 was written by SPL International, who also supplied a run-time support package to interface between RTL/2 and RSX-11M. [7] The SPL interface has several shortcomings [8], such as poor naming of the interface procedures, insecure use of system traps, inadequate error reporting and large memory overhead on file I/O.

Therefore, it was decided to use an alternative run-time support package called MTSLIB [9] which was developed at AECI and corrects most of the deficiencies of the SPL interface.

1.4.2 Multi-user Access to the Media Interface

1.4.2.1 The Media Data-base -

In order to keep careful control over access to the interface, user applications software is not given direct access to the serial line connected to the Media interface. Instead, a copy of all information relevant to Media is kept in a resident common data-base in the memory of the PDP-11 host computer. This data-base is called MEDCOM (for "Media common") and contains all the input and output data, times of last update, access control codes, setpoints, etc.

1.4.2.2 The Media Update Task -

A task called MEDUPDAT (for "Media update task") runs continually and updates the database regularly with fresh information from Media and writes output data from the data-base to Media. The update task is the only task that communicates directly with the serial link to Media; all other tasks, including user application programs, read and write data from/to the data-base and rely on the updating task to ensure that the data in the data-base is up-to-date and that the output data is passed on to Media.

1.4.2.3 The Media User Interface Library -

In addition to being denied direct access to the serial link, applications software is not given direct access to the data-base MEDCOM itself, but must access it in a controlled fashion, to ensure that application tasks cannot interfere with one another. To this end, a library of routines (called MEDUSER, for "Media user routines") provides easy read and write access to the data-base, and hence via the updating task to the Media interface itself. From the user point of view, accessing Media is a simple and direct operation, because the routines in MEDUSER make all of the following transparent:

1. Securing and releasing of the data-base for indivisible operations (see chapter 6)

2. The operation of the Media update task, and

3. Encoding and decoding of frames in the Micro-Media protocol and the sending and receiving of data link frames.

For example, to write the data OUTDATA to analogue output number 2, the user would simply write

```
WRMEDOUT( ANALOG, 2, OUTDATA);
```

and would not have to worry about securing, releasing, protocols, etc. The disadvantage is obviously the time uncertainty in controlling Media; i.e. one cannot know exactly when the data written to MEDCOM will reach Media because a variable amount of time may elapse before MEDUPDAT performs the transaction with Media.

1.4.2.4 Exclusive access to Media outputs -

Access to MEDCOM via the MEDUSER interface has been controlled in such a way that any user is permitted to read data from any part of the data-base, since this will not interfere with other users' activities, but a user can only write to those parts of the data-base to which he has "attached" before running his tasks. Before running any task which writes information to MEDCOM, the user must attach to the outputs involved by running a task called ATTACH, after which he has exclusive access to the outputs to which he has attached. The user may if he wishes specify to the ATTACH task which outputs to which he wishes to attach, or may allow ATTACH to give him whichever happen to be free.

A procedure is supplied in the MEDUSER interface which enables a user-written task to determine which outputs it is attached to. This approach allows user programs to be independent of which particular outputs are used, obviously a desirable feature since the availability of different outputs will clearly vary from time to time due to the muti-user teaching and research use. The ATTACH task is described in detail in chapter 7.

1.4.2.5 The Data-base Manager -

A task called MEDRMD (for "Media RMDEMO") provides a dynamically updated display of the activity of the data-base. This display is updated every second and gives details of which user is attached to each output, how much time has elapsed since the last update and which task is currently securing the data-base. This task is similar in spirit to the RSX-11 RMDEMO task [10] which gives a dynamic display of the use of the computer system resources such as memory, pool, task lists and disc space. Use of MEDRMD enables a system manager to monitor the activity of the data-base, and correct any complications that occur (such as users attaching to outputs and failing to detach afterwards). It also enables a user

to quickly and easily see whether MEDUPDAT is in fact updating the data base, and see to which outputs he is attached.

One of the accounts (account number [300,1]) is designated the data-base manager's account : a user with [300,1] as his protection UIC has more privileged access to the data-base than do the other users. By using the information from MEDRMD, he can keep control over the data-base: for example, he can forcibly detach from outputs users who fail to do so themselves.

1.4.3 The Necessity for Simulation

Although the primary function of this system for the purposes of teaching is to give the students hands-on experience of the control of a real process, practical factors dictate that much of the time the students will not in fact be controlling a real process, but instead a software simulation of the process. For example, there may be only one set of apparatus, or the apparatus may be delicate and easily damaged by a program which is not fully debugged. The data-base MEDCOM, therefore, contains not only all the data needed for the Media interface itself, but also an additional part which is used for simulation. This part of MEDCOM is not updated by MEDUPDAT, but instead by a simulation task which simulates the effect of the Media update task plus the Media interface and the process apparatus. From ther users' point of view, however, it is exactly as if he were controlling a real system. The only difference to the user is that when he attaches to outputs using ATTACH, he specifies that he wishes to use a simulated system.

To enable the easy simulation of analogue systems, a simulation package (see chapter 9) has been written which enables easy implementation on the digital computer of almost any analogue block diagram.

1.4.4 Graphics Interface

A library of routines simplifying the use of the Tektronix 4010 plotting terminal is provided. This library is not intended to be a complete graphics interface, but merely provides sufficiently many software building blocks to enable users to easily build up their own routines such as graph-plotting routines. This allows users to plot graphs of the parameters of the processes that they are controlling - for example the value of one of the analogue inputs against time. Hard copies of the graphs plotted may be obtained using the hard-copy unit attached to the plotting terminal.

1.4.5 System Management and Diagnostic Tools

Various diagnostic programs have been written which can be used to test the Media interface and the various software modules and libraries. These are described in chapter 8.

1.5 Typical Applications

1.5.1 Teaching

The system was used during the 1983 academic year to teach the fundamentals of real-time software and multitasking to final-year Electrical Engineering undergraduates at UCT. The project, which is described in detail in chapter 10, involved controlling the

temperature of air emerging from a 60 cm vertical column, as in figure 1-2:

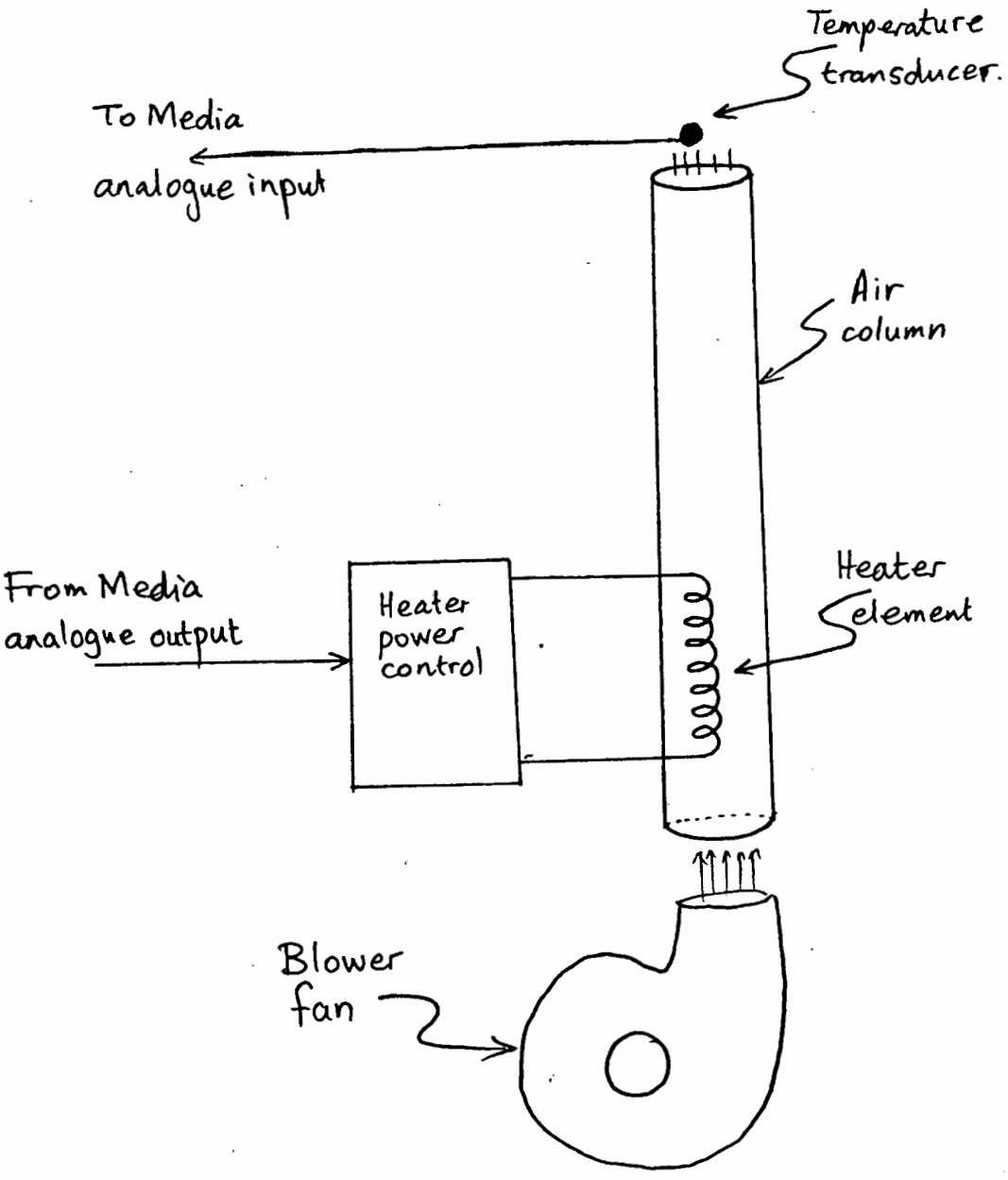


Figure 1-2: Air-column apparatus

Air is blown upwards at a constant rate by the blower fan and the temperature read by the thermistor is read in via one of the analogue inputs of the Media interface. The power output to the heater is controlled via one of the analogue outputs.

In addition, a simulation task could be run which simulated the operation of four such columns using the simulation portion of MEDCOM. The students' software, which accessed the data-base using the MEDUSER library, could be debugged and tested on the simulation system and then run on the real system - thus allowing up to five students to run their software simultaneously.

1.5.2 Research

A good example of a research project that could be undertaken using Media is the determination, using adaptive control techniques, of the characteristics of a variable reflux-ratio fractional distillation tower, illustrated in figure 1-3. Two analogue inputs would be used to measure the temperature at two points in the column, and another would give a measure of the purity of the emerging distillate. One analogue output would vary the power to the heater, and another would vary the reflux ratio.

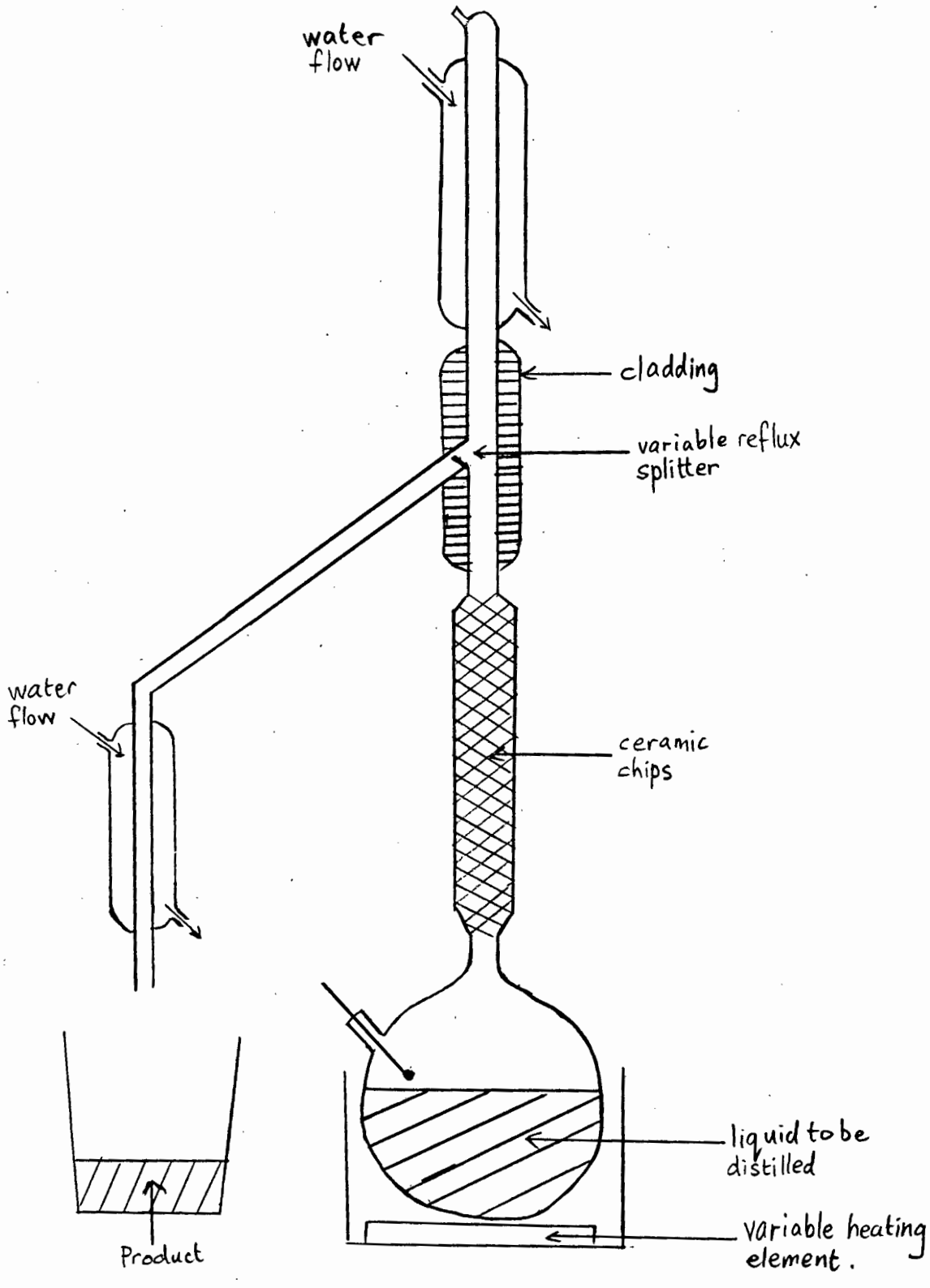


Fig 1-3: Variable Reflux-ratio distillation tower

CHAPTER 2

MEDIA

2.1 HARDWARE

2.1.1 Introduction

Media, or Modular Electronic Digital Instrumentation Assemblies [1], is a family of process control interfaces designed by Imperial Chemical Industries (ICI). The use of Media offers the following advantages:

1. Media provides "data highways" which give read, write and control access at memory speeds to any part of the system. It is therefore particularly suited to the control of more complex systems.

2. The highways provide random access to data.

3. Media is modular, since it is divided into groups of 19-inch card racks into which a wide variety of available

Media cards may be plugged. Because of this modularity, Media offers flexibility in configuration and capabilities.

4. Media can safely be used in a hostile electrical environment, since isolation is provided on all inputs and outputs.

Media systems come in three basic models:

1. Media Major.

   This is the largest of the family and is used to control large industrial plants involving several thousand plant signals.

2. Media Minor.

   This model is of intermediate size and fills the gap between Media Major and Micro-Media.

3. Micro-Media.

   Micro-Media is used to control small systems, typically consisting of up to 32 control loops.

The process data is transferred on the Media highways, of which there are two types:

1. Active highway. This contains those Media cards which can take charge of the highway as master.

2. Passive highway. The passive highway is controlled by the active highway and contains those Media cards which respond to commands but do not take control of the highway, e.g. digital inputs and outputs.

Up to 16 Media cards can be plugged into each 19-inch rack-mounting "Media bin". These bins can be of three types:

1. Active bins: These contain cards which are situated on the active highway.

2. Passive bins: These contain cards which are situated on the passive highway.

3. Mini-bins: Mini-bins have both an active and a passive part, and are only used in small Media systems.

2.1.2 The UCT Micro-Media installation

The UCT Media system [11], which is a Micro-Media system supplied by GEC, has a mini-bin and a passive bin housed in a welded open-frame chassis. The analogue and digital input and output lines from the Media cards are connected to three connector rails at the back of the Media chassis. A diagram of the connector rails is given in Appendix A.

The mini-bin has the following cards installed:

1. M7100 8085 CPU card

2. M7101 CPU-to-Media interface card

3. M6310 serial I/O card

4. M0410 minibin control card

5. M7000 PDP-11 interface (data)

6. M7001 PDP-11 interface (control)

7. M7002 LSI/Media bus interface with M7022 LSI-to Media cableform.

8. M0601 Highway terminator

The passive bin has the following cards installed:

1. Two M2001 16-line digital input cards. These 32 digital lines are intended to be driven by switch contacts, but have been brought out to the front panel (see appendix A) in such a way as to enable them also to be driven by TTL logic levels.

2. One M3302 16-line digital output card. These 16 lines emerge from the card with logic high being 24 volts, but have been brought to the panel (see appendix A) in such a way as to allow a choice between 24 volts and TTL logic levels.

3. One M3004 4-channel analogue output card. The outputs have 8-bit precision, with 00H being equivalent to -4mA, and FFH to -20mA. They have been brought to the panel in such a way as that they are also available as -1V to -5V voltage outputs (see appendix A).

4. One M1601 16-way analogue multiplexer card. Selection of a particular channel is described in Appendix B.

5. One M1000 10-bit analogue-to-digital converter card. The analogue input to this card is the one selected by the multiplexer card. The input voltage range is -1V (for 000H) to -5V (for 3FFH). The format of the data read is described in Appendix B.

6. M0003 passive highway terminator card.

A serious problem with the system is that the ground reference level for the digital cards is 15 volts below that of the analogue

cards. This means that there must be two ground levels (analogue ground and digital ground), which leads to unnecessarily complicated circuitry when interfacing Media to the circuit being controlled.

Specific details of the programming of each individual Media card are given in appendix B.

2.2 SOFTWARE

2.2.1 Outline

As was menioned in the introduction, the heart of the Micro-Media system consists of three boards: an 8085 CPU board,a Media interface board and a serial I/O board. These boards are plugged into the active section of the mini-bin. Two ROM chips on the CPU board contain a program, written in Coral-66, allowing Micro-Media to be used as a remote data input/output station, connected to the host computer via a serial data link. A simple half-duplex link-level protocol [12], described below, is implemented whereby the host sends command frames, and receives reply frames from Micro-Media.

The Micro-Media software continually scans the Multiplexed analogue inputs and stores the values read into an area of RAM called the "analogue list". The data items stored in the analogue list are called "list items".

The 32 bits in the two "digital change words" stored in RAM correspond to up to 32 16-line digital input cards. Whenever the state of any of the input lines to one of the digital input cards changes, the Media program sets the bit in the digital change

words corresponding to this card. The digital change words are cleared by the Media program after the host reads them. The philosophy behind the use of the digital change words is to allow the host to determine whether or not any of the digital input lines have changed state, without having to access each digital input card. In the present system, however, which has only two digital input cards, only two bits of the digital change words are used and no real advantage is gained over simply accessing both digital input cards.

Each Media card is identified by its "Media address", i.e. its address on the Media highway, and each item in the analogue list has a "list address". Media and list addresses are 10-bit addresses. In the LSI-11 Media system, the Media addresses correspond to successive locations in the I/O space of the LSI, as explained in appendix B.

The Micro-Media protocol is a half-duplex link-level protocol. All transactions between Media and the host are initiated by the host. The host sends a frame in which is encoded a command field to indicate to Media which of sixteen possible operations it must perform, and Media replies with a reply frame including a status field indicating whether or not the operation was successful, and if not, what went wrong. Frame data integrity is checked using a block check character. Error correction is not performed - the host is expected to take corrective action, such as retransmitting a frame, if the reply from Media indicated that the received frame was faulty. A timeout is implemented by Media whereby the time interval between any two successive bytes within a frame may not exceed a certain amount, or else the frame is rejected.

2.2.2 Media protocol frame format

Each frame consists of a number of bytes into which the various frame fields are encoded. The first and last byte of each frame

has bit 6 set (regarding the least significant bit as bit 0, and the most significant bit as bit 7). No other byte within a frame has bit 6 set. After the first byte of a frame is detected, all further bytes are included in the frame until another byte with bit 6 set is detected. This byte is then taken as the last byte in the frame.

This scheme has the disadvantage that the same method (setting bit 6) is used to indicate the start and the end of the frame, and therefore, should a start or end byte be lost, all start bytes will be identified as stop bytes (and vice-versa) and so no further frames will be transmitted correctly.

The following fields are present in all frames:

1. The command code. The command code informs the Media program which of sixteen possible operations (such as read digital change words or write to Media address) it should perform. The command code consists of bits 0 to 3 of the first byte in the frame. All possible command codes are listed in Table 2-1 and described below.

2. The terminal address. Bits 4 and 5 of the first two bytes contain a 4-bit terminal address, used to identify a particular terminal if a multidrop line is used. In the LSI-11 Media system, these bits are ignored.

3. The block check character (BCC). Bits 0 to 5 of the final byte in the frame form a BCC, which is formed as the logical exclusive-OR (vertical parity) of all previous bytes in the frame.

4. Parity. Bit 7 of each byte may be used as a parity bit. The LSI-11 Media system does not use the parity bit, but instead puts zero into these fields.

5. Reply status. In the case of Media-to-host reply frames, bits 0 to 3 of the second byte give status information as listed in Table 2-2. The status information informs the

2.2.3 Media operations

The operations which Media can perform in response to a command frame from the host are summarised in Table 2-1.

When data is written to Media, this is done in a two-stage process, so as to ensure integrity. A frame with a command field indicating a write (code C or D) is sent to Media, containing an item of data to be written to a specified Media or list address. Before taking any action, Media reflects the entire frame back to the host. If the host is satisfied with this reply, it then sends a confirmatory frame with a command field indicating "go" (code E) causing the data to be written, and Media replies with a 3-byte frame containing the status field. Should a confirmatory "go" frame not be received from the host, Media will not write the data, but will simply perform whatever operation is specified by the command field of the next frame received (i.e. the initial "write" frame will have had no effect).

The following list describes all the operations which may be specified by a frame from the host:

1. Single read from Media address (code 1). Media returns the value read from the specified Media address.

2. Single read from list address (code 2). Media returns the value read from the specified list address.

3. Block read from Media addresses (code 3). The frame from the host indicates how many items must be read, and the Media address of the first item. The items are read from consecutive Media addresses and returned inside the reply frame.

4. Block read from list addresses (code 4). Same as for (3), except that list addresses are used.

5. Read digital change words (code 7). The reply frame from

Media contains the digital change words.

6. Read the printer status word (code 8). The reply from Media contains the printer status word, which indicates the status of the printer local to the Media interface. Since our system does not have a local printer, this function is not implemented in the LSI-11 Media system.

7. Write to a single Media address (code C). The frame from the host contains the Media address and the data to be written to that address. Media does not actually execute the write until it receives a confirmatory "go" frame, but instead reflects the entire frame back to the host for checking.

8. Write to a single list address (code D). As for writing to a Media address except that a list address is used.

9. Go (code E). This is the confirmatory command used in the second stage of a write operation and informs Media that it may now execute the write specified in the previous host command frame.

10. Write text to printer (code 0). The frame received by Media contains text which must be printed on the printer local to the Media interface. Since our system does not have such a printer, this command was not implemented in the LSI-11 Media software.

CHAPTER 3

THE LSI-11 MEDIA SYSTEM

3.1 THE LSI-11 MEDIA SOFTWARE

3.1.1 General

This software component emulates the operation of the GEC proprietary Micro-Media software. Since the target LSI-11 microcomputer consists only of CPU board, memory and serial I/O interface, the operating system used must be built together with the Micro-Media software into a single task (called SMT.TSK) before being downloaded into the LSI-11.

The operating system used is SMT (for "Standard Multi-Tasking") version 18 which is a small (about 6K words) multitasking operating system designed for use in real-time control applications [4]. Because of its small size, SMT is ideally suited to applications such as this one where the system can be

compiled on a minicomputer and downloaded into a microcomputer such as the LSI-11 used. SMT is written mostly in RTL/2, but has assembler code inserts where necessary. The use of a high-level language makes the operating system easier to understand and to update.

SMT provides the framework into which the user may insert up to 16 tasks written in RTL/2 for his particular application. Full RTL/2 system and stream I/O facilities are provided for the tasks, as well as 16 user events and 16 facilities. SMT allows 255 levels of priority for the tasks. Standard RTL/2 error handling mechanisms are provided.

3.1.2 SMT System Modules

The operating system itself consists of the following modules:

3.1.2.1 The SMTB1 and SMTB1X modules -

These contain all the machine-dependent system code, including interrupt handling, real-time clock handling, unrecoverable error handling and system start-up code. Both are supplied with SMT, but SMTB1X is the one that must be used with the RSX-11 macro-assembler, since it contains certain .PSECT directives that the assembler requires.

Because SMT was originally written to run on the old pre-LSI PDP minicomputers, it does not exploit the larger instruction set of the LSI-11 to the full. The LSI-11 has additional instructions for processor status word access which were not present in the instruction sets of the earlier PDPs. To enable these more efficient instructions to be used, SMTB1X must be edited to change

the old instructions to the new ones. All the necessary changes are included in a file called SMTB1X.EDT (not supplied with SMT). An edited file, SMTB1XUP.RTL, is produced by the DEC editor, as follows:

```
EDT SMTB1XUP.RTL=SMTB1X.RTL
*INC =X /FI:SMTB1X.EDT
*XEQ =X
*EX
```

SMTB1X is compiled using

```
RTL SMTB1X,SMTB1X=SMTB1XUP/CN:F/PE:SMTB1X/TI:SMTB1XUP
MAC SMTB1X=SMTB1X
```

The /CN:F switch is needed because SMTB1X contains code inserts, the /PE switch is used to generate a .PSECT directive at the beginning of the output macro-11 source and the /TI: switch specifies a title.

3.1.2.2 The SMTU1 and SMTU1X modules -

These contain the machine-dependent user task definitions and initialisation code, i.e. the details about all the user tasks that will be built into the system - what the names for the base procedures of the user tasks are, what the tasks' priorities are, whether they are to be initialised into the running or suspended states, etc. Therefore, SMTU1X must be edited to configure it to run the two user tasks SMTCOMS and SMTMULTI (these are described later). This module as supplied also includes the device driver routines for the console serial line of the LSI. However, in this application, these have been replaced by custom-written drivers in the SMTDEVDRV module (below), and so must be deleted from this module.

As for the SMTB1X module above, SMTU1X is the one that must be

used with the RSX-11 assembler. A file, SMTU1X.EDT, is provided which contains all the necessary commands for the DEC editor to edit SMTU1X.RTL to produce SMTU1XUP.RTL.

SMTU1X is compiled using

```
RTL SMTU1X,SMTU1X=SMTU1XUP/CN:F/PE:SMTU1X/TI:SMTU1X
MAC SMTU1X=SMTU1X
```

The meaning of the switches used was described in the section on SBTB1X.

3.1.2.3 The SMTB2 module -

This module contains the machine-independent system routines. Included amongst these are routines connected with event flags such as WAIT, DELAY, SET and RESET, the securing and releasing of facilities, the printing of error messages when unrecoverable errors occur and the standard RTL/2 stream I/O procedures such as IREAD and TWRT.

The only modification necessary is the suppressing of the output of the error-message printing routine ERPRIN (for reasons explained later). This is done by commenting out the bulk of the routine. Examination of the listing of the routine will show that an alternative method would have been to set the error count ECT to zero just before ERPRIN tests its value.

SMTB2 is compiled using

```
RTL SMTB2,SMTB2=SMTB2/TI:SMTB2
MAC SMTB2=SMTB2
```

3.1.2.4 The RTLCTL module -

This module contains the control routines for RTL/2 tasks running under SMT. These include the trap handlers, array bound checking routines, stack unwind routines, type conversion and all arithmetic routines such as compare, add, subtract, multiply, divide and arithmetic shift.

The module is written in macro-11 and does not need to be modified. It is assembled using

```
MAC RTLCTL=RTLCTL
```

3.1.2.5 The SMTB3 module -

This contains the system data tables and the stacks for the tasks which form part of the SMT operating system itself (e.g. the real-time clock task). The contents of the system data tables is not of any relevance from the user point of view.

No modifications are needed to SMTB3, and it is compiled using

```
RTL SMTB3,SMTB3=SMTB3/TI:SMTB3
MAC SMTB3=SMTB3
```

3.1.3 User modules

3.1.3.1 The SMT device driver SMTDEVDRV -

The device drivers supplied with SMT as part of the SMTU1X module are not suitable for use as drivers of the Media serial line, which must transmit and receive 8-bit binary data without special significance to any of the characters.

The supplied drivers were deleted and a module SMTDEVDRV was written to replace them. The module consists of the hardware input and output console line interrupt servicing routines as well as byte input and output procedures intended to be assigned to the standard RTL/2 stream input and output procedure variables IN and OUT in the SVC data brick

```
SVC DATA RRSIO;
  PROC () BYTE IN;
  PROC (BYTE) OUT;
ENDDATA;
```

The description below of these four procedures should be read in conjunction with the listing of SMTDEVDRV:

1. PROC INSERIAL ().

   This is the hardware serial input interrupt service routine and is vectored to via location 000060. It is executed whenever a byte has been received on the line and is ready for processing by the Media software.

   Inspection of the listing will show that INSERIAL takes this byte and puts it into a buffer (INAREA.INBUFF) and increments the buffer pointer (INAREA.INPT). It then queues the console input event INEVENT and returns.

2. PROC OUTSERIAL ();

   This is the hardware serial output interrupt service routine and is vectored to via location 000064. It is

executed on completion of the transmission of a serial output byte.

All it does is to queue the printer event flag PREV to indicate to the software that transmission of the character is complete.

3. ENT PROC INBYTE () BYTE;

This procedure, together with INSERIAL, replaces the SMT-supplied console input procedure INTTY. It is assigned to the standard RTL/2 byte input procedure variable IN by the Micro-Media communications task SMTCOMS. All 8 bits of all input bytes are passed, none of them being given any special treatment.

Its operation is as follows. If there are bytes left in the input buffer, the output pointer INAREA.OUTPT is incremented and the byte pointed to is temporarily saved in the variable CHAR. A test is then done to see if the byte is the last one left in the buffer. If it is, then the buffer pointers INAREA.INPT and INAREA.OUTPT are reset to zero. A suspension of interrupts using HLOCK takes place during this critical test-and-reset operation. The procedure then returns the byte saved in CHAR.

4. ENT PROC OUTBYTE (BYTE B);

This procedure is used to output bytes to the console line. It is assigned to the standard RTL/2 byte output procedure variable OUT by the Micro-Media communications task SMTCOMS. All 8 bits of all bytes are passed, and no special significance is attached to any of the characters.

SMTDEVDRV is compiled using

```
RTL SMTDEVDRV,SMTDEVDRV=SMTDEVDRV/CN:F/PS:DEVDRV/TI:SMTDEVDRV
MAC SMTDEVDRV=SMTDEVDRV
```

3.1.3.2 The SMTCOMS user task -

3.1.3.2.1 General description -

SMTCOMS is the serial I/O communications task. It performs the following operations:

1. Receiving and transmitting of all data link frames.

2. Decoding the Micro-Media protocol serial link command frames received from the host, and detecting any errors in these frames (such as faulty block check character).

3. Constructing the Micro-Media reply frames which it will send to the host.

4. Performing all operations (such as Media read or write) that are specified by the command frames received from the host.

3.1.3.2.2 Detailed description -

This description should be read in conjunction with the program listing.

SMTCOMS is activated every time an input byte from the host arrives down the serial link. It waits for the event flag INEVENT which is set by the input interrupt servicing routine INSERIAL.

The input bytes are built up into complete frames (a frame being a complete command from the host in the Media protocol format). This is done by the procedure BUILD() which is called for each input byte.

Checking of the input characters is only done once the input frame is complete. The successive bytes of an input frame must arrive within two seconds of each other, otherwise the frame is rejected.

The start and end bytes of each frame are detected by testing bit 6 of each incoming byte - a byte in a Media frame has bit 6 set if and only if it is a start or end byte.

When the BUILD() routine indicates that the frame is complete, PROCESS() is called to process the frame. It checks that the block check character (BCC) is correct, decodes the frame, takes appropriate action according to the Media code, builds up a reply frame and outputs the reply down the console line.

The fundamental procedures for accessing Media highway addresses are WMEDIA and RMEDIA. These are also called by the SMTMULTI task when it accesses Media.

1. ENT PROC RMEDIA (INT AD) INT. The returned integer contains the data read from the card at Media address AD. AD is multiplied by 2 and added to the base address 176400 to convert it to an LSI address.

2. ENT PROC WMEDIA (INT ADDR, VALUE). The data VALUE is written to the Media address ADDR.

Should a Media access error cause a bus time-out unrecoverable error, control is passed to the unrecoverable error processing label UNRECOV. Unrecoverable error processing is discussed in section 3.1.4.

Procedure GETADD extracts the Media or list address from the address field of input frame, and DECODE similarly extracts the data from the data field.

3.1.3.2.3 Error conditions recognised -

The SMTCOMS task recognises several error conditions :

1. LENERR : Length of input frame is inconsistent with the Media code used.

2. TOOMANY : Too many bytes in input frame - input buffer overflow.

3. TIMEERR : Timeout error while waiting for input.

4. BADCODE : Invalid Media code received.

5. BLOCKERR : BCC of received message incorrect.

6. MEDIAERR : Media access error.

7. UNEXPGO : GO code received which was not preceded by a write message.

The Micro-Media protocol, however, only allows five types of error status to be returned :

1. Parity, framing or overrun error in input frame. This is returned for LENERR and TOOMANY.

2. Timeout on character input. This is returned for TIMEERR.

3. Invalid Media code. This is returned for BADCODE and UNEXPGO.

4. BCC faulty. This is returned for BLOCKERR.

5. Media error. This is returned for MEDIAERR.

3.1.3.2.4 Compiling SMTCOMS -

SMTCOMS is compiled using

```
RTL SMTCOMS,SMTCOMS=SMTCOMS/CN:F/TI:SMTCOMS
MAC SMTCOMS=SMTCOMS
```

3.1.3.3 The SMTMULTI user task. -

SMTMULTI is the analogue multiplexer scan task. Its function is twofold:

1. It must continually read the digital input cards and update the digital change words in RAM accordingly.

2. It must continually read the multiplexed analogue input data and store this in the analogue list in RAM.

Every 60 milliseconds it reads in the digital inputs, compares them with the values read the previous time, and stores the digital change words accordingly. It then writes to the multiplexer to select the next analogue input to be read, waits 40 to 60 milliseconds to allow the multiplexer to settle, and reads in the data from the ADC card. Only one analogue input is read each time. It therefore takes 16 times as long, i.e. about one second, to scan all 16 inputs.

Any Media access which causes a bus timeout on access is regarded as a Media error, and causes error processing to pass to the unrecoverable error label. The unrecoverable error processing then puts the card responsible for the error out of scan.

A list of cards which are out of scan is maintained, as well as a pointer to this list. Each time round the loop, one of the out-of-scan cards (if there are any) is accessed to see whether it has come back into scan, and if so, its element is removed from the out-of-scan list. If however, the card is still out of scan, an unrecoverable bus time-out error will again occur. It is for this reason that only one of the out-of-scan cards can be tested at a time.

SMTMULTI is compiled using

```
RTL SMTMULTI,SMTMULTI=SMTMULTI/CN:F/TI:SMTMULTI
MAC SMTMULTI=SMTMULTI
```

```
SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC (INT) ERP;
ENDDATA;
```

3.1.5 Building SMT.TSK

Once all the modules have been compiled as described already, they may be task-built to form SMT.TSK using the following command

```
TKB @TKBSMT
```

where TKBSMT.CMD is

```
SMT/-HD/-MM,SMT/-SP/MA/CR=LOWCR,SMTB1X,SMTB2,SMTB3
RTLCTL,SMTU1X,SMTDEVDRV,SMTMULTI,SMTCOMS
/
STACK=0
UNITS=0
PAR=EXEPAR:0:160000
/
```

LOWCR.MAC reserves space for the bottom area of LSI memory which will contain the interrupt vectors. It consists simply of

```
.PSECT
.BLKW   128.      ; Reserve 256. bytes for int. vectors
.END
```

3.1.6 Note on debugging the user SMT modules.

In the SMTCOMS, SMTMULTI and SMTDEVDRV modules, certain RTL/2 lines have been commented out using triple percent signs ("%%%"). These lines were originally present while the software was being

debugged, and cause messages to be printed down the console line describing what the programs are doing. If it is ever necessary to debug or extend these routines, the abovementioned debugging lines can be temporarily put back by changing all occurrences of "%%%" to "%%". Once debugging is complete, changing "%%" to "%%%" will cause the debugging lines to be commented out again.

3.2 REMOTE SYSTEM DOWNLOADING AND STARTUP

3.2.1 The ODT program

3.2.1.1 General -

The remote system start-up procedure is structured around the use of a program called ODT, so called because it interacts with the microcoded ODT (Octal Debugging Technique) program [13] which runs on the LSI-11 when it is halted.

ODT, which is run on the host and provides semi-transparent access to the console line of the remote LSI-11, enables the operator to perform any of the operations that are supported by the LSI-11's console ODT. ODT can therefore be used to examine and change any of the LSI's memory and register locations. As was mentioned in the introductory chapter, ODT (together with the bootstrap loader SMTLOAD) also provides the means whereby the LSI-11 Micro-Media software SMT.TSK can be downloaded into the remote LSI-11 and started running.

Ideally, ODT should provide full transparency between the

operator's terminal on the host and the LSI-11's console line, so that from the operator's point of view it would seem that his terminal was directly plugged into the LSI. Such transparency can be achieved using RSX-11M's asynchronous system traps (ASTs) [14] to deal with the incoming characters from the two ports, but the simpler approach of using standard QIO calls was used which allowed the program to be made operational sooner and with less effort.

Unfortunately, use of the RSX-11M QIO$S [15] I/O directive only allows one to achieve partial transparency. The operator types in a complete ODT command followed by a carriage return. This command is scanned and checked by the program, and if it is a satisfactory ODT command, is sent down the line to the LSI. The program then waits for a reply from the LSI, in most cases with a one-second time-out. The reply received is then sent to the user's terminal.

This approach requires the program to calculate the number of characters expected in the reply from the LSI, and in order to do this, a record is kept of what "state" the LSI is currently in. The four states used are :

1. START : The LSI is waiting for a command and has output an '@' prompt.

2. MEMORY : An LSI memory location is currently open.

3. REGISTER : An LSI register location is open.

4. PSW : The LSI processor status word location is open.

If the user types in a command which is inappropriate for the current state of the LSI, it is rejected and an error message is displayed.

3.2.1.2 The flush (F) command -

The flush ('F') command, which is recognised in all states, is provided so that the user can flush the input type-ahead buffer of the host serial port to get rid of any unwanted input characters. The program issues an I/O directive to read all characters without echoing them, with a one-second time-out (a QIO$S #IO.RAL!TF.RNE directive). Any input in the buffer, plus any that arrives in the next second, is discarded after being output to the user's screen. It is recommended that the flush operation be done on program startup and also whenever replies from the LSI appear not to be properly aligned with the commands sent out. The flush command also returns the program to the START state.

3.2.1.3 Commands recognised in START mode -

In addition to the flush command, the following are recognised in START mode :

1. Open memory location
   <octaldigitlist><'/'> (e.g. '700000/')

   Memory location 'octaldigitlist' is opened and the contents displayed. 'Octaldigitlist' must consint of one to six octal characters. The program is put into MEMORY mode.

2. Open register location
   <'$'|'R'><octaldigit><'/'> (e.g. 'R6/')

   Register number 'octaldigit' is opened and the contents displayed. the program enters REGISTER mode.

3. Open PSW location
   <'$'|'R'><'S/'> (i.e. '$S/' or 'RS/')

   The PSW is opened and the contents displayed. the program enters PSW mode.

4. Go
   <octaldigitlist><'G'> (e.g. '1070G')

   Processing is started from the location specified by 'octaldigitlist'. No timeout is put into effect and the program will wait until the LSI has output 11 characters (normally caused by the LSI halting again). The program remains in START mode.

5. Proceed
   <'P'> (i.e. 'P')

   Causes processing of the LSI to continue from the location pointed to by the value of register 7. As for the Go command, no timeout is imposed and the program remains in the START state.

6. Help
   <'H'|'HELP'>

   Sends to the terminal a short summary of the operation of the ODT program.

7. Exit
   < control-Z>

   Stops execution of the program and returns control to MCR.

8. Load
   <'L'> (i.e. 'L')

   This command is described in the section on downloading SMT (below).

3.2.2 Commands recognised in MEMORY and REGISTER state

1. Next location
   [<octaldigitlist>]<'N'> (e.g. '1000N')

   The contents of the current location are replaced by 'digitlist' if this field is present. Then the next location is opened and its contents displayed. The state remains unchanged.

2. Close location
   [<octaldigitlist'>]<'C'> (e.g. '34C')

   The contents of the current location are replaced by 'octaldigitlist' if this field is present. Then the location is closed and the program enters START state.

3.2.2.1 Commands recognised in PSW mode -

In PSW mode, only the Close location command is recognised. Its syntax and operation is the same as for MEMORY/REGISTER mode.

3.2.2.2 Downloading the LSI-11 Media software -

The 'L' command, which can be invoked in START state, is used to download a bootstrap loader into the LSI, cause this downloaded loader to run and download the file 'SMT.TSK' into the LSI.

On typing 'L' the user is asked

DOWNLOAD THE ABSOLUTE LOADER (Y/N) ?

If the answer is Y, the absolute loader, stored in an octal dump format in the file SMTLOAD.ABS, is loaded, word by word, into the LSI, using the standard ODT commands, as if a person had typed it all in by hand. All the information loaded is also displayed on the terminal screen. When this is complete, the program asks,

LOAD SMT (Y/N) ?

If Y is typed, then the file SMT.TSK is loaded in 512-byte blocks down the line. After each block, the LSI returns a checksum byte, which the ODT program compares to the checksum it calculated. If a checksum error occurs, it is necessary to restart the LSI, flush the input buffer ('F' command) and start loading again. On the other hand, if SMT is successfully loaded, the message 'LOADING COMPLETE' is displayed and the program asks

START SMT RUNNING (Y/N) ?

If Y is typed, execution of the SMT program is started, and the LSI-11 Media system is functional. The user will normally then type control-Z to exit to MCR.

If N is typed, then the ODT commands can be used to examine and modify the downloaded SMT program.

Step-by-step instructions for Media system startup are given in appendix M.

3.2.2.3 Terminal settings -

The PDP port currently being used as the Media serial link is TT10:. For the program to download correctly, it is essential that this terminal be set to read-pass-all, as follows :

```
SET /RPA=TT10:
```

It is also recommended that the port be set to SLAVE, so that it ignores unsolicited input, and to NOECHO so that if a spurious character occurs, it will not be echoed back and forth across the link :

```
SET /SLAVE=TT10:
SET /NOECHO=TT10:
```

The link speed is 9600 baud :

```
SET /SPEED=TT10:9600:9600
```

These four commands are included in DL0:[1,2]STARTUP.CMD and are executed as part of the host system start-up.

3.2.3 SMTLOAD - a bootstrap loader for the LSI-11

3.2.3.1 Description -

SMTLOAD is a 160-byte-long bootstrap loader for the LSI-11, which is put into the LSI memory by the ODT task using the 'L' command as explained above, and which loads the LSI-11 Media program SMT.TSK into the LSI memory.

SMTLOAD itself occupies the 160 bytes from address 157000 to 157240. It is loaded by the ODT program while the LSI is halted and in console-ODT mode. The file SMTLOAD.ABS, which is an octal dump of SMTLOAD, is sent down the serial line byte-for-byte as if a user had typed it all in by hand.

The ODT program then sets the PSW of the LSI to 000340 (priority 7) so that it cannot be interrupted, and starts the LSI running from location 157000.

The first two bytes received by STMLOAD are interpreted as the length in bytes of the file to follow. A running checksum is also maintained. After every 512 bytes received, the checksum byte is output down the console line for checking by the ODT program. Once all the bytes have been received, SMTLOAD halts.

SMT.TSK is then ready to be run by a "0G" command from ODT.

3.2.3.2 Constructing SMTLOAD.ABS -

SMTLOAD is assembled and taskbuilt as follows:

```
MAC SMTLOAD=SMTLOAD
TKB @TKBSMTLD
```

where TKBSMTLD.CMD is

```
SMTLOAD/-MM/-HD,SMTLOAD/-SP=SMTLOAD
/
STACK=0
PAR=EXEPAR:157000:1000
//
```

The resultant task is a binary file SMTLOAD.TSK which has to be translated into an octal dump format by the DMP utility :

```
RUN $DMP
DMP>SMTLOAD.ABS=SMTLOAD.TSK/BL:3
```

Then SMTLOAD.ABS must be edited (using EDT) as follows :

1. Delete the first two lines. These contain page heading information.

2. Delete all line-feed characters, by typing

   ```
   S/<LF>// %BE:%E
   ```

   where <LF> is the line-feed character on the terminal keyboard.

3. The last several lines will contain only zero data. Delete these lines.

CHAPTER 4

THE SERIAL LINK INTERFACE HARDWARE

4.1 The serial lines

There are three 4-wire links running between the computer rooms of the Electrical Engineering and Chemical Engineering buildings, a distance of some 600 metres. These lines were installed by the Post Office.

Two of these are currently in use - one as the line connecting the LSI-11 (in Chem Eng) to the host PDP-11/23 (in Elec Eng), and the other as a general-purpose terminal link to the PDP. Both run at 9600 baud and use the balanced-line interface units described below.

4.2 The serial interface units

Each unit contains the circuitry necessary to interface two independent RS-232C channels to two balanced 4-wire lines (i.e. the GPO lines).

A full cicuit diagram of the unit is given in Appendix D.

The chips used are the DS8820 dual differential line-driver and the DS8830 dual differential line receivers [16].

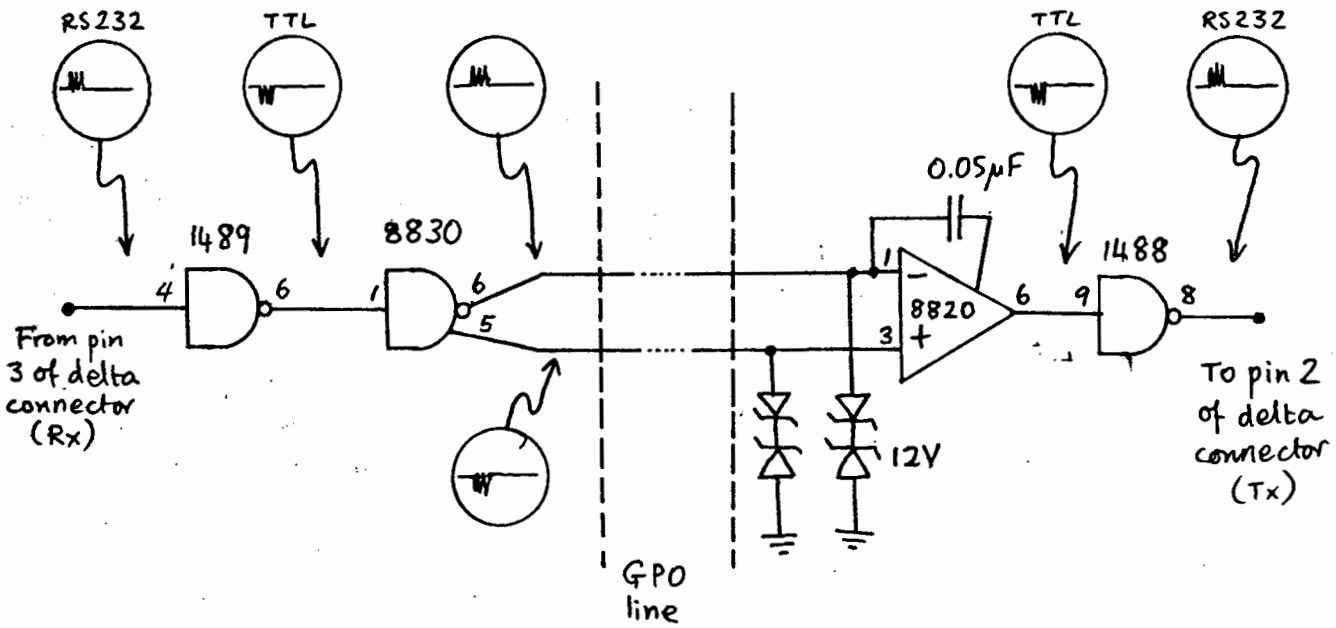


Figure 4-1.

Figure 4-1 shows the system circuit diagram for one of the two channels.

4.2.1 Circuit operation

Let us suppose that a terminal is plugged into the RS-232 connector for channel 1 of the unit. When a character is typed, the RS-232 level signal appears at pin 4 of the 1489. The 1489 converts this signal to an inverted TTL-level signal at pin 6, which drives pin 1 of the 8830. The 8830 converts this to a differential signal (the two levels being about +1V and +4V) between its AND and NAND outputs (pins 6 and 5). The AND output signal is inverted relative to the original input from the terminal, whereas the NAND output signal is not.

These outputs go via two of the wires in the GPO link to pins 1 and 3 of the 8820 in the interface unit on the other side of the link. These signals are limited between -12V and +12V by the protection zener diodes shown.

The line is terminated with a capacitor of 0,05 microfarads between pins 1 and 2 of the 8820. The capacitor is chosen such that the time-constant RC formed with the 170 ohm termination resistance R internal to the 8820 chip is three times [17] the line signal-propagation time. This time was measured as being about 3,6 microseconds.
Hence RC = 3 *3,6 microseconds, where R = 170 ohms, and from this we calculate C = 0,05 microfarads.

The 8820 converts the differential signal to a TTL-level signal inverted with respect to the original signal. This is converted to an RS-232 signal via the 1488 chip and appears on pin 8 of the chip, from where it is connected to the RS-232 delta connector.

CHAPTER 5

MEDCOM - THE DATA-BASE

5.1 Description

All the information obtained from Media is stored in a data-base called MEDCOM, which is continually being updated by the Media update task. MEDCOM is an RSX-11M resident common which is permanently in memory, and is linked to at task-build time by tasks that must access Media. User tasks wishing to access Media do this indirectly by reading or writing to MEDCOM using the user interface routines in the MEDUSER library (see chapter 7).

In addition to containing all information relevant to Media, MEDCOM also contains information which is used, as described in the introduction, to enable real-time simulation of systems. MEDCOM contains sufficient extra space to allow for the simulation of four analogue outputs, four analogue inputs, 16 digital output lines and 32 digital input lines. The simulation part of MEDCOM is not updated by the Media update task, but instead by a simulation task which simulates the effect of the Media update

task communicating with a Media interface connected to a process. The reason why simulation is necessary is given in the introductory chapter, and an example of a simulation task is described in the concluding chapter.

The basic structural item of MEDCOM is the MEDCARD (for "Media card") record, which is defined as

```
MODE MEDCARD (INT STAT, MEDDAT, ADDR, REAL SCANTIME);
```

where

1. STAT is a status word, the bits defined as follows:

```
Bit 0:  1 = card is in scan
        0 = card is out of scan
Bit 1:  1 = 'ADDR' field is a Media address
        0 = 'ADDR' field is a list address
Bit 2:  1 = output card
        0 = input card
Bit 3:  1 = analogue card
        0 = digital card
Others: Reserved
```

2. MEDDAT . For an input, MEDDAT is the data as read from Media by the the update task. For an output, MEDDAT is the data that the update task must write to Media.

3. ADDR is the Media or list address of the card (or item).

4. SCANTIME is the time (in seconds past midnight) of the most recent successful transaction between Media and the data-base for the item. (i.e. when last it was updated).

In the case of the digital inputs and outputs, the MEDCARD record corresponds to an actual card physically plugged into the Media

highway. The MEDDAT field then contains the state of sixteen digital lines.

However, a MEDCARD record does not directly represent a card on the Media highway in the following cases:

1. The analogue inputs are represented in MEDCOM by an array of 16 MEDCARDs, corresponding to the 16 analogue list items in RAM inside Media.

2. MEDCOM has four MEDCARDs to represent the four analogue output channels, even though the four outputs are physically situated on the same Media card.

3. The digital change words and the Media status word (see appendix E) are also represented by MEDCARDS.

4. Parts of MEDCOM which are used for simulation clearly do not correspond to physical Media cards.

The record used for describing an analogue output is the AOREC record, defined as follows:

```
MODE AOREC( INT UIC, USERINT, REF MEDCARD ANIN, REAL
 SETP);
```

where

1. UIC is the protection user identification code of the user attached to the output. This is used by the MEDUSER library routines to determine whether or not the calling program should be permitted to write to the corresponding output.

2. USERINT is an integer which is intended to be used for inter-task communication by the user attached to the corresponding analogue output.

3. ANIN is a pointer to the MEDCARD record representing the analogue input being controlled by this output.

4. SETP is the setpoint of this analogue input. The ANIN and SETP fields are provided to simplify the programming of single-input single-output (SISO) control loops since these are the most frequently encountered. If the system is not a SISO control loop, the ANIN and SETP fields are not used.

MEDCOM consists of two ENT data bricks:

```
ENT DATA INAREA;
  ARRAY(20) MEDCARD ANINP;       % 20 analogue inputs     %
                                 % (16 real, 4 simulated)%
  ARRAY(4)  MEDCARD DIGINP;      % 4 digital input cards  %
                                 % (2 real, 2 simulated)  %
  MEDCARD   MEDSTAT;             % Media status word      %
  ARRAY(2)  MEDCARD DIGCHAN;     % digital change words   %
ENDDATA;

ENT DATA OUTAREA;
  ARRAY(8)  MEDCARD ANOUTP;      % 8 analogue outputs     %
                                 % (4 real, 4 simulated)  %
  ARRAY(2)  MEDCARD DIGOUT;      % 2 dig. output cards    %
                                 % (1 real, 1 simulated)  %
  ARRAY(8)  AOREC   AODESC;      % descriptions of a.o.'s%
  ARRAY(2,16) INT DIGUICS;       % UIC's for each d.o.    %
ENDDATA;
```

We see that the MEDCARDs used for simulation are stored in the same arrays as those used for access to the real Media system. The real inputs and outputs always are the lowest-numbered elements in these arrays, and the simulation inputs and outputs are the higher-numbered elements. For example, the 16 real

analogue inputs are represented by elements 1 to 16 of the array ANINP above; and the 4 simulation analogue inputs are represented by ANINP(17) to ANINP(20).

INAREA holds all information that is read in from Media: a MEDCARD record for each of the 16 analogue inputs and the two digital input cards, MEDCARD records for the digital change words and the Media status word. For simulation, there are 4 MEDCARDs for the analogue inputs and 2 for the digital inputs.

OUTAREA holds all information about Media outputs: a MEDCARD record for each of the 4 analogue outputs and the digital output card, an AOREC record to describe each analogue output, and a UIC to control write access to each digital output line. For simulation, there are four MEDCARDs and four AORECS for the analogue outputs, one MEDCARD for the digital outputs, and a UIC for each digital output line.

5.2 Modifications needed for future changes

The numbers of real and simulation inputs and outputs of each type (analogue or digital) are given in LET statements in the RTL/2 source for MEDCOM and the other modules which refer directly to the data bricks INAREA and OUTAREA. Should it be decided to include further Media or simulation inputs or outputs at a later stage, it will be necessary to change these LET statements to reflect these additions to the system. This also applies, of course, should cards be removed instead of added. No further changes to the software should be necessary, except that MEDRMD (see chapter 7) would have to be rewritten because the layout of its screen display obviously depends on the number of inputs and outputs in the system.

5.3 Building MEDCOM as a resident common

This is described in appendix H.

CHAPTER 6

THE SOFTWARE INTERFACE BETWEEN MEDIA AND THE DATA-BASE

6.1 THE MEDIA UPDATE TASK

This task, called MEDUPDAT, runs continually and executes all the transactions between MEDCOM and Media. As explained in the introductory chapter, user tasks access Media indirectly by writing to and reading from MEDCOM, and rely upon MEDUPDAT to do the following at frequent intervals:

1. Write to Media the output data put into MEDCOM by the user tasks.

2. Read data from all Media inputs and put this data into MEDCOM for access by the user tasks.

Whenever an item of data is successfully transferred between Media and MEDCOM, the time of day is written into the relevant MEDCARD.SCANTIME field. This field allows user tasks to determine how much time has elapsed since the most recent update of the

corresponding input or output.

Each update of the data-base is done in the following order (the MEDLNK routines referred to are described in the section on MEDLNK below):

1. Analogue inputs 1 to 8 and then 9 to 16 are read from Media using the MEDLNK routine BLOCKIN to read in a block of list items. This read is done in two blocks because the 36-byte type-ahead buffer used by the RSX-11M full-duplex terminal driver is not large enough to receive the size of reply frame that results from a 16-item block read.

2. The first and then the second 16-line digital input card are read from Media, using the MEDLNK routine SINGLIN to do a single Media read.

3. The digital change words are read, using the MEDLNK routine GETDCW (get digital change words).

4. The four analogue outputs are written to in sequence, using the MEDLNK routine WRITE to write the data from MEDCOM to the Media address of the analogue output card.

5. The 16-line digital output card is written to, using the MEDLNK routine WRITE to write the data from MEDCOM to the Media address of the card.

If any transaction with Media results in Media indicating that a "Media error" has occurred, then the Media card in question is put out of scan (in/out of scan is indicated by one of the bits of the STAT field of a MEDCARD record). A card which is out of scan ceases to be written to or read from by MEDUPDAT, because a Media error usually means that the card is not present on the Media highway. However, after every ten complete updates of MEDCOM, the

update task tries to access those cards which are out of scan. If a further Media error does not occur, then the card in question is put back into scan.

The speed at which MEDUPDAT runs is not dependent on the frequency of access to MEDCOM by users, since it continually updates all data-base information. Rather, it is dependent largely on the overall load on the PDP-11, the serial link data transmission rate and the priority at which MEDUPDAT is running. MEDUPDAT pauses for 0.1 seconds after each complete scan of the data-base.

Each update involves the transfer of 189 bytes on the serial link, which amounts to a total time of about 0.2 seconds per update at 9600 baud. This figure constitutes the limit to the speed of MEDUPDAT when the PDP-11 is not loaded, but under normal loads the processing time limits the update rate to about once per second. It was found that with a priority of 80, MEDUPDAT was capable of updating at least every two seconds, even when the system was heavily loaded.

Thus, the interface can only be used to control systems which have time constants in excess of about four seconds. This constitutes a fairly serious limitation on the range of applications for which it may be used. Fortunately, the applications envisaged use systems with time-constants of the order of minutes rather than seconds.

6.2 PROCEDURES INTERFACING BETWEEN THE DATA LINK AND MEDCOM

The Media update task is the only task which accesses the serial link directly (with the exception of the diagnostics described in chapter 8). It does this by calling procedures within a library MEDLNK of routines which transfer data between Media and MEDCOM.

These routines in turn call a routine MESSANS, inside the module LINKLB, which controls the link itself.

6.2.1 The Serial Link Control Software

The link-control software is contained in the module LINKLB. This module contains only one ENT procedure,

```
ENT PROC MESSANS( REF ARRAY BYTE OUTBUF, INT OUTLEN,
                  REF ARRAY BYTE INBUF , INT INLEN  ) INT;
```

whose function is to output a frame of OUTLEN characters from the buffer OUTBUF down the serial link, and receive a reply frame consisting of INLEN characters into the buffer INBUF, giving error status information in the integer returned.

This procedure is the lowest-level procedure used on the host side of the link. All frames to Media are transmitted via a call to MESSANS.

The write is done using the RSX-11M QIOW directive [15]. The directive queues the buffer OUTBUF to the full-duplex terminal driver, which is instructed to write all characters (QIOW$S #IO.WAL), and a one-second timeout on output is imposed. This method enables the standard RSX-11M full-duplex terminal drivers to be used, and therefore avoids the necessity of writing a device driver for the host to handle the Media link.

Reading the reply is complicated by the fact that one ideally would want to read until the second occurrence of a character with bit 6 set, and this cannot be done directly in RSX-11M unless a special device driver to implement this is written. Therefore the number of characters expected in the reply (i.e. INLEN) is passed to MESSANS so that it knows how many characters to read. A

problem with this is that, should an error reply be received, its length will be three rather than the expected INLEN characters. If one were to issue a QIOW directive to the driver with (say) a three-second time-out to read all INLEN characters, one would discover that whenever any sort of error occurred, the time-out would occur, thus disabling all activity on the link for the full three-second time-out period.

Since this is not acceptable, what is done is to read the first three characters of the reply, to determine whether they form a complete reply (by checking whether bit 6 of the third byte is set), and if not, to read the remaining (INLEN - 3) characters, also using a QIOW directive. A 1-second timeout is imposed on the first read, and a 3-second time-out on the second read. The QIOW directives used instruct the terminal driver to read the input characters, without echoing them, and put them into the buffer INBUF (QIOW$S #IO.RAL!TF.RNE).

This method only works because all error replies are 3 characters long. Using the above method, time-outs on input should never occur, irrespective of the error encountered (short of failure of the line), and so maximum link bandwidth can be utilised.

Errors reported in the returned integer are:

1. Timeout error on input or output

2. QIO directive failure on input or output

3. INLEN or OUTLEN <= 0

6.2.2 MEDLNK: Procedures interfacing MEDCOM to the data link

The module MEDLNK consists of the procedures which are used (by the Media update task) to transfer data between MEDCOM and the link to the remote Media system.

MEDLNK is the module that does all the encoding into, and decoding from, the Media protocol. All error checking on reply messages from Media is also done here. MEDLNK has procedures enabling single or block read from a Media or list address, single write to a Media or list address, and read of the Media status word or the digital change words.

During critical activities, the data-base is secured using the secure and release procedures described later in this chapter. The period during which MEDCOM is secured has been made as short as possible.

If any access to Media fails for any reason other than a Media error, a second attempt is made. Should this fail too, the standard RTL/2 error recovery procedure ERP is called, causing an error message to be sent to the system console. This message gives an error number (see appendix L) as well as recording the time at which the error occurred.

Occurrence of a Media error causes the access to be aborted and the "out of scan" bit of the .STAT field of the relevant MEDCARD record is set, so that the Media update task can put the card out of scan.

The ENT procedures in the module MEDLNK are described below. In all cases, the SCANTIME element of each MEDCARD record involved is updated if and only if the transaction with Media was successful. The integer returned is the error status, which may indicate successful operation, Media error or one of several other errors (see appendix L for details). The Media code MCODE which

indicates to Media which operation must be performed (see Table 2-1) is not checked for validity.

1. ENT PROC SINGLIN (INT MCODE, REF MEDCARD INP) INT;

   This procedure does a single read from the Media or list address INP.ADDR, and the data, which is a 16-bit integer, is written into INP.MEDDAT.

2. ENT PROC BLOCKIN (INT MCODE, FIRST, LAST,
   REF ARRAY MEDCARD MBLOCK) INT;

   This procedure does a block read (Media or list depending on MCODE). The number of items read is LAST minus FIRST, and the Media/list addresses are MBLOCK(FIRST).ADDR to MBLOCK(LAST).ADDR. The values read are written into the MEDDAT fields of elements FIRST to LAST of the array MBLOCK. LAST must be greater than FIRST, but the procedure does not check this. However, it does check that the .ADDR fields in the array MBLOCK are suitably ordered.

3. ENT PROC WRITE(INT MCODE, REF MEDCARD OUTDAT) INT;

   This procedure writes the data OUTDAT.MEDDAT into Media or list address OUTDAT.ADDR.

4. ENT PROC GETDCW(REF ARRAY MEDCARD DIGCN) INT;

   This procedure gets the digital change words, using a block Media read, and puts them into the MEDDAT fields of DIGCN(1) and DIGCN(2).

6.3 SECURING AND RELEASING DEVICES AND FACILITIES

Two secure/release mechanisms are used - one for the serial data link and one for the data-base MEDCOM.

6.3.1 Securing and releasing the serial link

This is easily done because the MTSLIB RTL/2 interface provides procedures SECDEV (LUN) and RELDEV (LUN), where LUN = logical unit number of the device to be secured/released. These procedures are used in MESSANS to control access to the link.

Strictly speaking it is not necessary to secure and release the link because under normal circumstances only one task, namely MEDUPDAT, accesses the link. However, it is done because it is good programming practice to do so, and because it enables the Media link diagnostic tasks (see chapter 8) to be run while MEDUPDAT is active.

6.3.2 Securing and releasing the data-base

An altogether more thorny problem is that of securing and releasing the data-base MEDCOM, to ensure that indivisible operations such as the updating of a MEDCARD record cannot be interrupted by other tasks.

Securing of facilities is usually done using a "test-and-set lock". This involves performing an indivisible operation which sets a flag or location to a state representing "device secured" while at the same time determining whether or not the flag was set

before the operation. Using this mechanism, to secure a facility a program sets the flag to the "device secured" state, and if the flag was not in the "device secured" state before the operation, then the task knows that it has secured the facility. On the other hand, if the flag was already in the "device secured" state then some other task has secured the facility and the task must try again until such time as it succeeds.

Releasing the facility is done simply by setting the flag to some state other than "device secured".

The MTSLIB interface provides secure and release procedures [18] which use the group-global event flags as a test-and-set lock to secure and release facilities. The radix-50 name of the task currently securing each facility is stored in a data-brick inside RSXBA2, which must be installed in a common partition of memory, and hence any tasks using the secure and release must be linked to this partition at task-build time.

These MTSLIB procedures have certain disadvantages :

1. It turns out to be necessary, because of limitations on the operating system directives available, to have a secured device represented by a flag in the reset state, and a non-secured device represented by a flag in the set state. The problem with this is that at system start-up, all flags "wake up" in the reset state, thus representing all facilities secured. It is thus necessary at system start-up time, or at any rate before any securing or releasing is done, to explicitly set any event flags that will be used for facility locking.

2. An attempt to remedy the above complication led to the author of MTSLIB partially relaxing the test-and-set mechanism by allowing a task to secure the facility regardless of the state of the event flag, provided that the field in RSXBA2 containing the name of the task

currently securing the facility was filled with zeros, as it would be on system start-up.

Unfortunately, this made it possible for two tasks to be simultaneously secured to the same facility. The circumstances under which this occurs are the following. The release procedure first clears the task-name field in RSXBA2 to indicate that no task is currently secured to the facility and then releases the facility by setting the relevant event flag. If a second task attempts to secure the facility between these two operations, it succeeds because the task-name field is zero, and therefore both tasks are simultaneously secured to the facility. Worse still, the first task then almost immediately releases the facility by setting the event flag, leaving the the facility free to be secured by a third task!

Because of this, the secure/release procedures could not be trusted to work reliably.

3. If a task is aborted (by using the ABO command or signing off) while it is securing a facility, the facility is never released and so no other task can access the facility, which therefore "hangs". The facility is not automatically released on task abortion because the facility secure/release mechanism is not part of the RSX-11M operating system, which consequently cannot be expected to tidy up on behalf of a mechanism of which it has no knowledge.

4. SECURE and RELEASE use group-global event flags. This means that all users of the facility would have to be in the same UIC group or else the mechanism would have no value.

It was felt necessary to use global event flags, so as to make it possible for users of all UICs to use Media. This is especially the case because the system is designed to support different categories of user, viz. students and researchers, whose accounts will be assigned different UIC group codes. It was also necessary to correct the bug mentioned in (2) above. This was done by restoring the strict test-and-set lock mechanism and providing a task, MCOMINIT, which initialises the relevant event flags by setting them and is run automatically on system start-up by the system start-up file LB:[1,2]STARTUP.CMD.

A partial solution was also found to problem (3) above, which was to write a task specifically designed for aborting tasks using the Media data-base (see appendix F).

A description of all the procedures relating to securing and releasing as well as the modifications necessary to RSXBA2 are given in appendix G. Forming a common partition RTLLIB containing RSXBA2 and linking user programs to this partition are described in appendix H.

CHAPTER 7

USE AND MANAGEMENT OF THE MEDIA SYSTEM

7.1 USE OF THE SYSTEM

7.1.1 Control of an individual user's access to the system

7.1.1.1 Introduction -

Since one of the fundamental requirements of the system is that it must allow multi-user access in such a way that no users are permitted to interfere with one another, it is of primary importance that access to the data in MEDCOM be carefully controlled.

It was decided to allow each user read access to all of the data in MEDCOM, since this could not compromise the security of other

users' tasks. However, it is clear that a user must not be permitted to write data to parts of MEDCOM (such as outputs) that are in use by another user, since this would destructively interfere with his system. No harm is caused, however, by another user reading from any part of the data-base.

A user may wish to write to MEDCOM under the following circumstances (see the chapter on MEDCOM for a description of the data records referred to):

1. Writing data to a Media output. The data will have to be put into the MEDDAT field of the MEDCARD record corresponding to the required Media output.

2. Writing to the AODESC.USERINT integer associated with one of the analogue outputs (this integer being intended for a user's inter-task communication).

3. Assigning an analogue input as the one being controlled by a particular analogue output. This is used in controlling SISO systems and involves writing to the AODESC.ANIN field of the output in question.

4. Setting the setpoint of an analogue input when controlling a SISO system, by writing to the AODESC.SETP field associated with the corresponding analogue output.

Therefore we see that all write operations to MEDCOM involve writing to a MEDCARD or AODESC record which is associated with a particular output. Consequently, the need to control write access to MEDCOM reduces to controlling write access to the (analogue and digital) outputs.

7.1.1.2 The ATTACH task -

This control is effected by requiring that a user "attach" to the outputs he needs, using a task called ATTACH, before running his tasks, and detach from these outputs (also using ATTACH) after his work is completed so that the outputs are free for use by other users.

Since there are several users of Media, however, a user may not have the same output every time he uses Media - another user may be busy with the one that he had on the previous occasion. On the other hand, he may for some reason wish to use the same output every time (the ATTACH task allows a user to specify the particular output required if he wishes). In either case it would clearly not be satisfactory for him to have to modify his software to suit the particular outputs he happens to attach to. Therefore a means is provided (the ATTACHED routine in the MEDUSER library described below) whereby a user progam can determine which outputs the user is attached to.

Corresponding to each analogue and digital output, the ATTACH task maintains an integer in MEDCOM containing the protection UIC of the user who is attached to the output, and hence authorised to write to it. If no-one is attached to an output, then the integer is zero. Any attempt by a user of the MEDUSER procedures to write to a part of MEDCOM not corresponding to an output to which he is attached, will be aborted and the standard RTL/2 error recovery procedure ERP called. For the digital outputs, the corresponding attached-UIC integers are stored in the array DIGUICS in the brick OUTAREA of MEDCOM. For analogue output number I, the attached UIC is kept in AODESC(I).UIC.

The security of this mechanism is dependent upon all users accessing MEDCOM only via the "official" interface, namely MEDUSER. Therefore, users should not be given specifics of the exact layout of MEDCOM.

7.1.1.3 Using the ATTACH task -

Attaching to outputs is done by the user prior to running his control (user applications) task. The user runs the task ATTACH, and is asked whether he wishes to attach or detach. If the reply is attach, then he is given the following menu of choices:

1. Any digital output (Real Media)
2. Any digital output (Simulation)
3. Specific digital output
4. Any analogue output (Real Media)
5. Any analogue output (Simulation)
6. Specific analogue output

Note that it is during the attaching process that the user determines whether he will be using the real Media system or the simulation system.

Digital outputs 1 to 16 and analogue outputs 1 to 4 are the real Media outputs. Digital outputs 17 to 32 and analogue outputs 5 to 8 are the simulation outputs.

Choice (1), (2), (4) and (5) imply that the user wishes to attach to an output, but does not mind which one it is. The program will find the lowest-numbered output channel that is not yet attached to, attach the user to it and print out an appropriate message. If none are available, the program will inform him of this.

Choices (3) and (6) imply that the user wishes to attach to a particular output. If the output required is not already attached to, then the program attaches the user to it; otherwise it informs him that the output is not available.

In all cases, if the required output is out of scan, then the program will inform the user and not allow him to attach to it.

Once the user has finished using the output, he must detach himself from it so that it becomes available for use by other users. To do this, he runs ATTACH and requests the Detach menu, which is as follows:

1. Detach from digital output
2. Detach from analogue output
3. Detach from all outputs

If the choice is (1) or (2) the user is asked which output number is to be detached. The program will only allow a user to detach from those outputs to which he is attached. Choice (3) detaches the user from all outputs (analogue and digital) to which he is attached.

The ATTACH task secures and releases the data-base in such a way as to ensure a test-and-set lock mechanism on attach (this mechanism is described in chapter 6), which makes it impossible for two users both to be attached to the same output at the same time, even if they simultaneously request to attach.

7.1.2 Applications-user access to Media

7.1.2.1 Introduction -

Applications-user access to Media is done by calls to a library, called MEDUSER, of routines the aim of which is not only to ensure strict control over the data-base MEDCOM, but also to provide a simple and easy-to-use interface which makes it seem from the user point of view as if he is accessing Media directly. In reality, of course, all Media data passes via MEDCOM and the Media update task, but this occurs transparently to the user software.

The global event-flag secure and release procedures described in chapter 6 are used inside the MEDUSER routines to secure the data-base and thus prevent other tasks accessing the data-base during critical sections of the code when it is necessary to write or read data in an indivisible manner. The user never secures or releases MEDCOM explicitly. This minimises the chance of MEDCOM becoming "hung" by a user securing and not subsequently releasing it. This possibility is further discussed in the section below on aborting Media tasks.

The MEDUSER procedures allow the user to perform the following operations:

1. Determine which outputs the user is attached to (proc ATTACHED).

2. Write data to an analogue or digital output (proc WRMEDOUT).

3. Read data from an analogue or digital input or output (proc RDMEDIA). Reading from an output is done in a multi-tasking application when one task wishes to see what value another task has written to an output.

4. Read or write to one of the AODESC.USERINT integers which are intended for inter-task communication use (procs WRCOMMINT and RDCOMMINT).

5. Read the time that an input or output was last scanned by the Media update task or the simulation task (proc RDSCANTIME).

6. For a SISO system, create a logical association between an analogue output and an analogue input indicating that the input is being controlled by the output; or determine which input is being controlled by a given output (procs SETANINP and GETANINP).

7. For a SISO system, to set or read the setpoint of an analogue input being controlled by a given analogue output (procs SETSETPT and GETSETPT).

7.1.2.2 Using the routines in the MEDUSER library -

The user should include the following LET statements in his source program :

```
LET ANALOG  = 0;
LET DIGITAL = 1;

LET INPUT   = 0;
LET OUTPUT  = 1;
```

These values are passed in the ADSWITCH and IOSWITCH parameters of the procedure calls. (e.g. X := RDMEDIA(ANALOG, INPUT, 3) would read analogue input number 3 and put the value into X).

All errors that occur are reported via ERP or RRGEL. A list of error numbers is given in appendix L.

In the description of the procedures in MEDUSER, CHANNUM represents the channel number of the digital or analogue input or output channel in question. The following channel numbers are valid:

1. Digital inputs. Numbers 1..32 are the real Media digital input; numbers 33..64 are the simulation digital inputs.

2. Digital outputs. Numbers 1..16 are the real Media digital outputs; numbers 17..32 are the simulation digital outputs.

3. Analogue inputs. Numbers 1..16 are the real Media analogue inputs; numbers 17..20 are the simulation analogue inputs.

4. Analogue outputs. Numbers 1..4 are the real Media analogue outputs; numbers 5..8 are the simulation analogue outputs.

The following is a description of the procedures in the MEDUSER library.

1. ENT PROC ATTACHED (INT ADSWITCH, REF ARRAY INT OUTARRAY);

   This procedure examines the AODESC.UIC or DIGUICS fields in MEDCOM, and puts into the array OUTARRAY the channel numbers of those outputs to which the user is attached. The rest of OUTARRAY is filled with zeros. For example, if ADSWITCH = DIGITAL and the user is attached to digital outputs 1, 5 and 7 then the procedure will set OUTARRAY(1) to 1, OUTARRAY(2) to 5, OUTARRAY(3) to 7 and the other elements of OUTARRAY to zero. The length of OUTARRAY must be greater than or equal to the number of outputs attached to.

2. ENT PROC WRMEDOUT(INT ADSWITCH, MDATA, CHANNUM);

   This procedure writes the data MDATA to the analogue or digital output number CHANNUM. CHANNUM must be in the correct range. For an analogue output, only the least significant eight bits of MDATA are used because the DAC is 8-bit. The digital output value is 0 if MDATA is zero, or 1 if MDATA is not zero. In order for the write to succeed, the user must be attached to the output and the output must be in scan.

3. ENT PROC RDMEDIA(INT ADSWITCH, IOSWITCH, CHANNUM) INT;

This procedure reads the analogue or digital input or output number CHANNUM from MEDCOM and returns the value read. CHANNUM must be in the correct range. Digital inputs and outputs will be returned as 0 or 1. Analogue inputs (outputs) will be returned as a 10-bit (8-bit) number in the ten (eight) least significant bits of the returned integer and the other bits zero.

4. ENT PROC WRCOMMINT(INT CHANNUM, VALUE);

This procedure writes VALUE into AODESC(CHANNUM).USERINT which is an integer used for inter-task communication. CHANNUM must be in range 1..20 and analogue output number CHANNUM must be attached to.

5. ENT PROC RDCOMMINT(INT CHANNUM) INT;

This procedure reads the value of the inter-task communication integer corresponding to analogue output number CHANNUM and returns the value read. CHANNUM must be in the range 1..20.

6. ENT PROC SETANINP (INT ANOUTPNUM,ANINNUM);

This procedure causes analogue input number ANINNUM to be logically associated with analogue output number ANOUTNUM by setting AODESC(ANOUTNUM).ANIN to point to analogue input ANINNUM. This would then mean that analogue input ANINNUM is the one which is being controlled by analogue output ANOUTNUM. ANOUTNUM and ANINNUM must be in the correct ranges and analogue output number ANOUTPNUM must be attached to.

7. ENT PROC GETANINP (INT ANOUTPNUM) INT;

This procedure returns the channel number of the analogue input logically associated with analogue output ANOUTPNUM

should not be a problem.

Under conditions (1) or (2) above, the chance of MEDCOM "hanging" is relatively small because it is only secured for short periods of time. Nonetheless, it was felt that an alternative aborting task to the RSX-11M ABO command, which does not ensure that the task releases MEDCOM before it exits, should be written. This task is called ABM (for "abort Media").

ABM is a non-privileged task which should be installed as ...ABM and uses the same command-line format as ABO, i.e.

```
ABM TASKNAME
```

or,

```
ABM
```

where the taskname is assumed to be the default taskname for the terminal, e.g. "TT7" if invoved from terminal TT7:.

ABM should be installed at a high priority (say 150.) to ensure that it operates quickly even when the system is very heavily loaded.

ABM works by securing MEDCOM, issuing an abort directive to abort the task and then releasing MEDCOM. In this way it is assured that the task being aborted is not securing MEDCOM. A detailed description of ABM is given in appendix F.

7.2 MANAGEMENT OF THE MEDIA SYSTEM

7.2.1 Introduction

One of the accounts (account number [300,1]) is designated the "data base manager's account". The data-base manager is the user who has logged on under this account and has [300,1] as his protection UIC. The software system allows the data-base manager to perform certain operations that other users may not - he is able to "unhang" MEDCOM should it "hang" for any of the reasons given in the above section, and he can forcibly detach any user from any output should that user fail to do so himself.

7.2.2 The MEDRMD display of data-base activity

The RSX-11M RMDEMO task [10] gives a dynamic display of the use of the computer resources such as memory, pool, task lists and free disc space. Since it is updated every second, the RMDEMO display gives the viewer a good idea of what is happening in the system from moment to moment, and whether or not any of the critical system resources, such as system pool, are dropping below acceptable limits.

A task called MEDRMD (for "Media RMDEMO") was written which is intended to give the data-base manager a similar display, except that the activity of MEDCOM rather than the RSX-11M system is displayed.

MEDRMD displays information relating to the parts of MEDCOM associated either with the real Media interface or with the simulation system. When MEDRMD is running, typing an 'S' displays information about the simulation portion of MEDCOM, typing control-C or control-Z causes MEDRMD to terminate and typing any other character displays information about the real Media interface.

MEDRMD displays the following information :

1. The protection UIC of the user attached to each analogue and digital output.

2. The date and time.

3. The name of the task that is currently securing MEDCOM.

4. The time (in seconds) that has elapsed since MEDCOM was last updated by MEDUPDAT. This is split into five subfields :

   a. Digital inputs 1

   b. Digital inputs 2

   c. Digital outputs

   d. Analogue inputs

   e. Analogue outputs

   Should any of the cards be out of scan, the text "*OUT OF SCAN*" is displayed in the relevant field instead of the time since last update.

Although MEDRMD is intended primarily for use by the data-base manager, the display may be viewed by any user. Uses for the display are the following:

1. To see which output channels are attached to, and which are free.

2. To check that the update task is running properly - if it

is not, then the time since last update will continually increase.

3. To see which task is currently securing MEDCOM - for example, if the data-base manager [300,1] sees that the "Task currently securing MEDCOM" field does not change, he can run UNHANG (see below) to remedy the situation where a task has exited while secured to MEDCOM.

The display is terminated and the screen cleared, as in RMD, when the control-C or control-Z character is typed at the keyboard. MEDRMD achieves this by using the AST-routines in the module AST (see Appendix I).

7.2.3 Detachment by the data-base manager

The ATTACH task allows the data-base manager to detach any or all of the users who are attached to outputs. By examining the MEDRMD display, the manager can see which users have failed to detach from outputs after completing their work on Media, and then run ATTACH to forcibly detach such users so that the outputs are free for others to use.

7.2.4 The UNHANG task

If the data-base manager sees by examining the MEDRMD display that MEDCOM is secured by a task which has exited without releasing it (the symptom for this is that the "Task currently securing MEDCOM" field in the display does not change, he can "unhang" the data-base by running the UNHANG task.

UNHANG first checks the protection UIC of the invoking user to ensure that he is indeed the data-base manager, and then releases MEDCOM by calling FREEMEDCOM (see appendix G) and exits.

CHAPTER 8

SYSTEM DIAGNOSTICS

Diagnostics have been written to test the operation of various parts of the system:

1. The micro-Media data link protocol

2. The Media input and output hardware.

3. The software interfacing between Media and the data-base (i.e. MEDLNK)

4. The data-base MEDCOM.

5. The user interface library MEDUSER.

8.1 A diagnostic for the Media data-link protocol.

This task is called MEDFRAME and may be used to generate Media data-link protocol frames, send them down the serial link to

Media, receive the reply frame from Media and then display in octal on the operator's screen each byte of both frames. This enables the operator to determine whether the frame format of the responses from Media is correct.

MEDFRAME gives the operator a menu allowing him to generate frames corresponding to any of the following commands to Media:

1. Single Media or list read.

   MEDFRAME prompts the operator for which Media or list address is to be read.

2. Block Media or list read.

   MEDFRAME prompts the operator for the starting address and length of a contiguous block of Media or list items to be read.

3. Read digital change words.

4. Read Media status word (the Media status word is described in appendix D)

5. Write to Media or list address.

   This implements the first stage of a two-stage write. MEDFRAME prompts the operator for the data to be written and the Media or list address to which it must be written.

6. Go.

   This implements the second, or confirmatory, stage of a two-stage write.

7. Synchronise.

This causes a single byte with bit 6 set to be output to Media. It is included because the protocol cannot distinguish between the first and last byte in a frame because both are indicated by having bit 6 set. Should Media lose synchronism, this operation will restore it.

8.2 Diagnostics to test the Media input/output hardware.

8.2.1 The simulation package.

It will be explained in chapter 9 how the simulation package could be used as a diagnostic for the Media hardware, by simulating a SISO system in software and then controlling it via two pairs of Media analogue inputs and outputs.

8.2.2 The diagnostic MEDTEST.

MEDTEST can be used to test or exercise either the analogue or digital Media inputs and outputs. The description below presupposes familiarity with the Media front panel described in appendix A.

Two categories of digital tests can be performed:

1. Monitor switches.

   This provides a quick visual indication of whether the

digital inputs and outputs are functioning correctly. The operator chooses digital inputs 1-16 or 17-32 and wires these to the correspondingly numbered switches on the Media front panel. MEDTEST continually reads the digital inputs and writes the same data to the digital output lines, which are displayed on the LEDs on the panel. Altering the state of any of the switches will alter the state of the corresponding LEDs. The test is terminated by typing control-C at the keyboard.

2. Cyclic test.

The operator connects digital output lines 1-16 to the digital input lines, in one of the following formats:

1. Output lines 1-16 to input lines 1-16 (respectively).

2. Output lines 1-16 to input lines 17-32 (respectively).

3. Output line 1 to input lines 1 and 17, output line 2 to input lines 2 and 18, etc.

MEDTEST generates a series of constantly changing bit patterns in a 16-bit word. For each change, MEDTEST writes the 16 bits out to the digital outputs and then reads in the state of the digital input lines which the operator connected to the outputs. If the values read fail to match those written, a message to the operator's screen informs him of the error, giving the data value written and the incorrect value read back. After every sixteen write-read pairs, a '+' character is sent to the operator's screen. Typing control-C terminates the test.

For the analogue test, MEDTEST prompts for the numbers of four analogue inputs that it will associate with analogue outputs 1 to 4. In a continual loop, MEDTEST reads from these four analogue inputs and writes the data (scaled down from the 10-bit value read to an 8-bit value) to the analogue outputs. The four outputs thus will follow the voltages applied to the corresponding four analogue inputs. If this is not the case with one of the input-output pairs, then this pair is faulty. Typing control-C terminates the test.

8.3 A diagnostic for the Media-MEDCOM interface routines

The task MEDLNKTST provides a test for the routines in MEDLNK which interface between the serial link and MEDCOM.

MEDLNKTST is similar to the Media protocol diagnostic MEDFRAME, but works at a higher level. It uses the subroutines in MEDLNK to communicate with Media, and can be used to test any of the routines in MEDLNK (i.e. SINGLIN, BLOCKIN, WRITE, GETMED and GETDCW).

MEDLNKTST gives the operator a menu allowing the following choices:

1. Single Media or list read.

   The operator specifies the Media or list address of the item to be read. SINGLIN is then called to perform the read. The data read, and the error status returned by the call, is displayed.

2. Block Media or list read.

The operator specifies the starting address and number of items for a contiguous block of Media or list items to be read. BLOCKIN is then called to perform the read. The data read, and the error status returned by the call, is displayed.

3. Read digital change words.

The digital change words are read from Media by a call to GETDCW and displayed on the screen together with the error status returned by the call.

4. Read Media status word.

The Media status word (see appendix E) is read by calling GETMED and displayed on the screen together with the error status returned by the call.

5. Write to Media or list address.

The operator specifies the data and the Media or list address to which the data is to be written. A call to WRITE performs the write. The error status returned by the call is displayed.

8.4 Diagnostics for MEDCOM.

The task MEDRMD described earlier can also be used as a diagnostic for MEDCOM since it gives a running display of activity on MEDCOM.

8.5 Diagnostics for the user interface library.

8.5.1 The MEDUSER-testing task MUTEST

This task, called MUTEST, enables the testing of all of the subroutines in the MEDUSER library.

The operator is prompted with a menu consisting of the names of the routines in the library. After choosing one of these he is prompted for the values of each of the parameters that must be passed to the routine, e.g. channel number, data, etc. The appropriate MEDUSER routine is then called and any data returned by the routine is displayed on the screen. If an error occurs then an appropriate message is printed.

This task could also be used to inspect and change the data in MEDCOM (and hence Media) in an interactive way. For example, if a device controlled by one of the digital outputs for some reason must be turned off in a hurry, the quickest and simplest way to do this would be to write zero to the appropriate output using MUTEST.

8.5.2 The procedure READMEDCARD.

READMEDCARD is a procedure which was included in the user interface library MEDUSER to allow an indivisible read of a MEDCARD record in MEDCOM. It was written to assist in the debugging of MEDUSER.

```
ENT PROC READMEDCARD ( INT ADSWITCH, IOSWITCH, CARDNUM,
                       REF MEDCARD CARD);
```

This procedure reads the MEDCARD record determined by ADSWITCH, IOSWITCH and CARDNUM from MEDCOM and puts it into CARD. CARDNUM must be in the correct range (1..2 for digital output, 1..4 for digital input, 1..8 for analogue output and 1..20 for analogue input).

CHAPTER 9

A GENERAL-PURPOSE ANALOG SIMULATION PACKAGE

9.1 INTRODUCTION

The analogue simulation package was written with the following applications in mind:

1. It would facilitate the writing of simulation tasks which could simulate real processes interfaced to Media. Student tasks to control these processes could then be developed and debugged using the simulation part of MEDCOM and the simulation task, and when ready used on the real system.

2. It can be used within the student or research tasks which control the processes. For example, any analogue controller (a simple example is a proportional-plus-integral controller) could be implemented using the simulation package.

3. It can be used to test the operation of the Media interface hardware, without having to connect a process to Media.

   Consider, for example, the case where a control task has been developed to control a given SISO process. Suppose that it writes it outputs its control signal to analogue output number 1 and reads the process output from analogue input number 1.

   A second task is written using the simulation package to simulate the SISO process itself, reading its input from analogue input number 2 (say) and writing its output to analogue output number 2.

   If output 1 is connected to input 2, output 2 connected to input 1 and the control and simulation tasks executed, then the control task should control the simulated process. Failure to control correctly would indicate that one of the Media outputs or inputs used is not functioning correctly.

4. It can also be used for any other real-time or off-line simulation of analogue systems, possibly totally unrelated to Media.

In addition, a library of plotting subroutines is provided simplifying the use of the Tektronix 4010 plotting terminal. The description of this is included here because the plotting interface is intended chiefly to be used in conjunction with the simulation software.

Analog simulation languages, such as SIMC, MIMIC, and CSMP, are special-purpose computer languages enabling the user to simulate on the digital computer any system that can be represented on an analog computer. The digital computer has the advantage of a far

greater range and accuracy of real number representation, thereby avoiding the necessity for the tiresome process of scaling the inputs to integrators and amplifiers. In addition, the digital computer can easily perform any non-linear operation that may be required, which is not readily achievable on the analog computer.

During somewhat frustrating attempts to use the analog simulation languages SIMC and MIMIC on the Varian and Univac 1100/86 respectively, it occurred to the author that there would be several advantages in using a general-purpose high-level language such as RTL/2 rather than one of the dedicated analog simulation languages. A digital computer is, after all, capable of easily and accurately performing any of the operations of an analog computer, with the possible exception of integration. The main advantage of such an approach would be that one would not be constrained by the (usually rather narrow) restrictions placed on the user by simulation languages. Instead, one would be able to do anything permissible within the general-purpose language used, such as file manipulation or (in this application) calling the Media access routines in the module MEDUSER.

The main difficulty in developing the present package was to find a way of performing an accurate running integration on a variable, knowing only the present value f and past values $f_n, f_{n-1}, f_{n-2}, \ldots$ of the input variable f(t), and the past values $y_n, y_{n-1}, y_{n-2}, \ldots$ of the output variable y(t). Examples satisfying this requirement are the trapezoidal rule

$$y_{n+1} \equiv y_n + \int_{t_{n-1}}^{t_n} f\,dt = y_n + \tfrac{1}{2}\left(f_{n+1} + f_n\right)\Delta t$$

and the Simpson formula

$$y_{n+1} = y_n + \tfrac{1}{6}\left(f_{n+1} + 4f_n + f_{n-1}\right)\Delta t$$

where $\Delta t = t_{n+1} - t_n$.

However, these methods are not nearly as accurate as another

method, called the fourth-order Adams' method [19]:

$$y_{n+1} = y_n + \frac{1}{24}\left(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}\right)\Delta t$$

The advantage of this method is that the error is of the order of the time-step to the fifth power, being equal to

$$-\frac{19}{720}(\Delta t)^5 f^{(IV)}(\xi) \quad , \text{where } \xi \in (t_n, t_{n+1})$$

which is very small indeed.

The package implements an initialisation section, a main loop (each time round representing an increment of time DT) and a finishing-off or "tidying-up" section. The package can implement either real-time or offline simulation.

The functioning of the package is summarised in figure 9-1. The main procedure (RRJOB) is situated inside the package and is not written by the user. The user must supply the three routines SIMINIT(), SIMJOB() and SIMTIDYUP() shown in the figure. These are described in the next section.

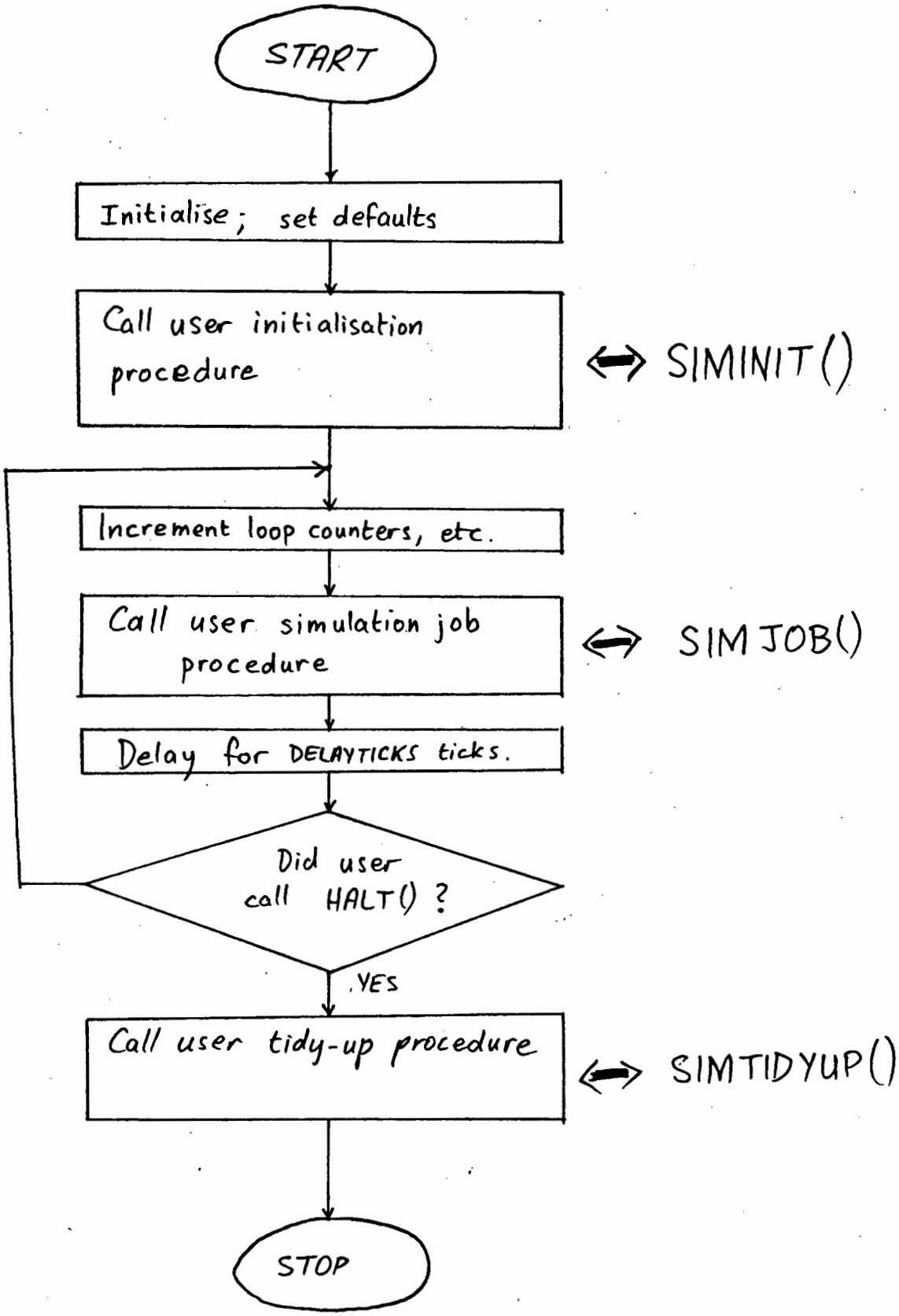


Figure 9-1 : Operation of the simulation package.

9.2 Using the simulation package

The user is required to do little more than draw the system block diagram and write down the system equations from this.

Three RTL/2 subroutines must be provided by the user:

a) ENT PROC SIMINIT()

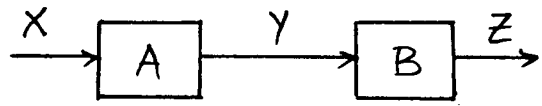
This procedure is called before the main loop and consists of any initialisation or change of default options (the defaults are real-time rather than off-line control, a delay of one second each time round the main control loop, and no AST processing). that the user may desire.

This routine may, for example, be used to read in the system constants of the analogue block diagram being simulated (such as time constants or gains, etc.) For this purpose, standard RTL/2 stream terminal I/O is initialised by the package before SIMINIT() is called.

Any other initialisation of any nature that must be performed before simulation is started (e.g. opening files) must be done in SIMINIT.

b) ENT PROC SIMJOB()

This procedure is called each time round the main loop and must contain the equations of the block diagram. The user must, however, be careful to get the equations in the right order. For example, if one were to simulate the following system



one could write

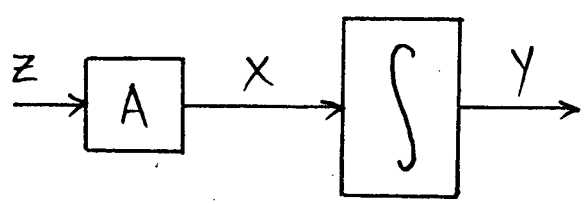
```
Y := X*A;
Z := Y*B;
```
(1)

or

```
Z := Y*B;
Y := X*A;
```
(2)

Obviously (1) is better than (2) because in (2) Z is calculated using the value of Y left over from the previous time round the loop, whereas in (1) the value of Y is first updated, then used.

All variables should be of type REAL except for inputs to integrators, which should be ARRAY(4) REAL for reasons given in the description of the INTEGRATE subroutine below. Elements 2 to 4 of this array can be ignored by the user because they are only used by the integration routine. The user should regard element 1 of the array as the actual variable. For example:



Here X is an ARRAY(4) REAL, Y is REAL and Z is REAL. The user writes

```
X(1) := A*Z;
INTEGRATE(X,Y,INITIALCOND);
```

The user can, of course, write to the screen or disc any information he chooses.

c) ENT PROC SIMTIDYUP()

This procedure is called after the main loop, and could be used to do any tidying-up needed, e.g. sending to the Tektronix screen information stored during the main loop, or closing files, etc.

The user has access to the following data brick used by the package:

```
ENT DATA SIMDATA;
  INT N,
      DELAYTICKS;
  REAL DT;
  ARRAY(4) REAL TIME;
ENDDATA;
```

These items are:

a) N:

The number of times round the main loop so far.

b) DELAYTICKS:

If simulation is real-time, this is the number of ticks (fiftieths of a second) of delay that will occur after each time round the loop. If some value other than the default value of 50 is required, this should be set up in SIMINIT(). If simulation is offline, DELAYTICKS is set to zero by the call to OFFLINE() (see below).

c) DT:

The time-step. For offline simulation, DT must be set up by the user in SIMINIT(). If simulation is real-time, DT is determined by the package as a running average over the past four sampling times - this is optimal for the integration method used. DT will then be equal to the execution time of

SIMJOB() plus DELAYTICKS ticks.

d) TIME:

If simulation is real-time, then this array will contain the actual real time (in seconds past midnight) of the present and past sampling instants.

The user can call the following procedures in the package:

a) ENT PROC() HALT

This procedure halts the simulation when called from SIMJOB(). The main loop is terminated on return from SIMJOB() and simulation falls through to SIMTIDYUP();

b) ENT PROC() OFFLINE

The package defaults to real-time simulation. If offline simulation is wanted, then OFFLINE() should be called from within SIMINIT(), in which case DT must also be set up by the user. A call to OFFLINE() sets DELAYTICKS to zero.

c) ENT PROC () NOAST

The package defaults to a mode whereby during the main loop processing may be interrupted at any time by typing control-C, causing an AST (Asynchronous System Trap) and the appearance on the screen of a menu of choices, e.g. whether to continue, restart, or abort. If this feature is not required, NOAST() should be called from within SIMINIT().

d) ENT PROC INTEGRATE (REF ARRAY REAL XDOT, REF REAL X,
REAL X0)

This procedure is the heart of the package and performs the integration. The input variable XDOT must be an array of

four reals. This is because the integration algorithm must be able to access past values of the input variable XDOT. For each integrator used, an array of four reals to represent the input variable must be declared by the user.

These past values of the input variable are stored as a stack consisting of XDOT(2), XDOT(3), and XDOT(4). This stack is maintained by the subroutine and the user need only consider XDOT(1) as the input variable. Each time INTEGRATE is called it updates the past values in this stack by "bumping" them down by one:

```
XDOT(4) := XDOT(3); % update oldest past value.,   %
XDOT(3) := XDOT(2);
XDOT(2) := XDOT(1); % update newest past value     %
                    % with present value of XDOT. %
```

The output variable X must be a real variable and NOT an expression.

X0 is the "initial condition" value for the integrator. Although it is only needed on the first call to the subroutine, it is passed every time. This approach, which leads to a user program layout similar to that used in most simulation languages, avoids the need to write another routine whose sole function is to initialise integrators.

9.3 Linking to the package

The user module containing the three user-written routines is linked to the package by setting up a taskbuild command file (say SIMTKB.CMD) containing the following:

```
USERMODULENAME=USERMODULENAME
SIMBGS,SCREEN,AST
@COMINTFCE
/
STACK=300
ASG=TI:1:2
LIBR=RTLLIB:RW
COMMON=MEDCOM:RW
//
```

The user then types TKB @SIMTKB and runs the resultant task.

9.4 PLOTTING LIBRARY FOR THE TEKTRONIX 4010 TERMINAL

The module PLOTLIB contains subroutines simplifying the use of the Tektronix 4010-1 plotting terminal for graphics output. This library is not intended to be a complete graphics interface, but merely provides sufficiently many software building blocks to enable users to easily build up their own routines such as graph-plotting routines. This allows users to plot graphs of the parameters of the processes that they are controlling - for example the value of one of the analogue inputs against time. Hard copies of the graphs plotted may be obtained using the hard-copy unit attached to the plotting terminal.

The Tektronix terminal was previously used with the Electrical Engineering department's Varian minicomputer, and was connected

via a current-loop serial line. The line-driver hardware of the terminal was converted to produce RS-232/C levels at 9600 baud so that it could be plugged into a serial port of the PDP-11/23.

The Tektronix screen has a resolution of 1024 points (horizontal) by 781 points (vertical). All graphics is done by means of escape-sequences as described in [20].

PLOTLIB contains the following ENT data brick:

```
ENT DATA PLOTDATA;
    REAL XOFFSET, XFACTOR, YOFFSET, YFACTOR;
    REAL XMIN := -1.0, XMAX := 1.0,
         YMIN := -1.0, YMAX := 1.0;
    ARRAY (20) BYTE REPLYBUF;
ENDDATA;
```

where

1. XMIN, XMAX, YMIN and YMAX are the X and Y values of the edges of the screen in the units used by the user.

2. XFACTOR and YFACTOR are scaling factors to convert real numbers in the ranges XMIN to XMAX and YMIN to YMAX respectively to integers in the ranges 0 to 1024 and 0 to 781 respectively.

3. XOFFSET and YOFFSET are offsets used in the above conversion.

The following ENT procedures may be used :

1. ENT PROC SCALE (REAL XMN, XMX, YMN, YMX);

   This procedure defines the limits of the screen as

called. Both axes will then range from -1.0 to 1.0.

3. ENT PROC GRAPHMODE ();

The terminal is put into graphics mode, so that it will be ready for graphics commands like MOVE and DRAW.

4. ENT PROC ALPHAMODE ();

The terminal is put into alphanumeric mode, so that it is ready to output or input alphanumeric characters.

5. ENT PROC DRAW (REAL XPOS, YPOS);

If the terminal is in graphics mode, a line is drawn from the current position on the screen to the point at (XPOS, YPOS). If the point is off the edge of the screen, ERP is called and the offending coordinate is truncated to the appropriate limit (XMAX, XMIN, YMAX or YMIN).

6. ENT PROC MOVE (REAL XPOS, YPOS);

If the terminal is in graphics mode, the current screen position moves to (XPOS, YPOS) but no line is drawn. Points off the edge of the screen are treated as in DRAW.

7. ENT PROC CLEARSCREEN ();

The entire screen is cleared of all alphanumerics and graphics. A delay of one second is executed to wait for the terminal to complete the screen-clearing process.

8. ENT PROC CROSSHAIRS ();

In graphics mode, this procedure causes a pair of crosshairs to appear on the screen. The position of the crosshairs is adjusted using the two potentiometers

alongside the keyboard.

9. ENT PROC GETCROSSHAIRS (REF REAL XPOS,YPOS);

   The current position on the screen of the intersection point of the crosshairs is written into XPOS and YPOS.

An example of the use of these routines is the program STAR.TSK which plots a pattern of lines on the screen. The task expects the Tektronix to be connected to TT12:.

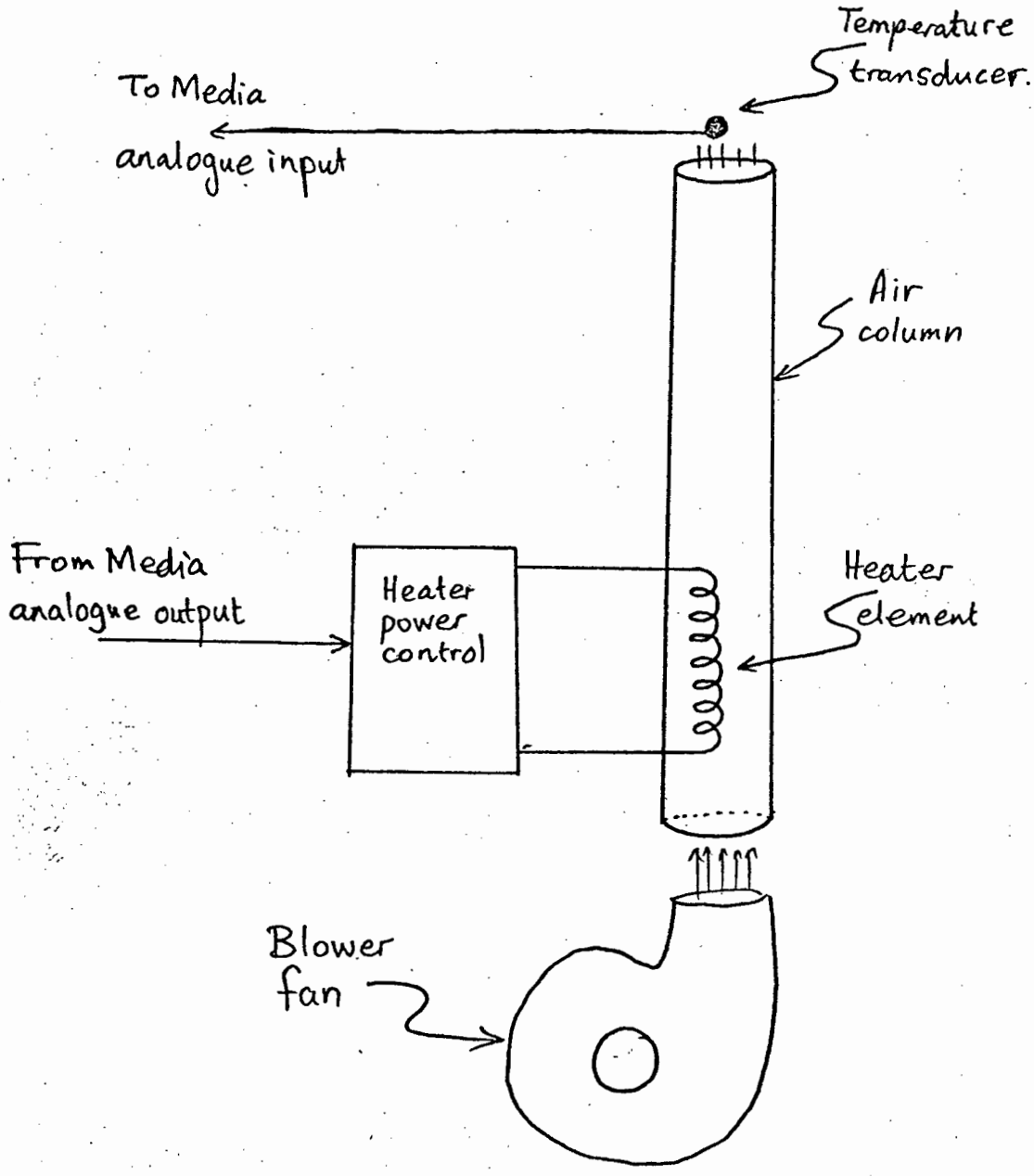
CHAPTER 10

CONCLUSION

A fitting conclusion to this thesis is to give an example of how the system has been used in practice. To this end, a description is given of the real-time project given to Electrical Engineering final-year students at UCT during the second trimester of 1983; this project having involved extensive use of all aspects of the Media system and the associated software.

The class consisted of about 40 students and the project was done in groups of two students each. The project instruction sheet is included as appendix K.

The apparatus used , developed in the Chemical Engineering department, is an 18-inch vertical thermally insulated column of air as illustrated below. The blower fan shown blows air up the column at a constant rate, and the temperature of the air emerging from the top of the column is measured by the thermistor whose output signal is connected to one of the Media analogue inputs. A heater element inside the column heats the air as it travels up the column. The heater can supply 0 to 1000 watts of power. The circuitry controlling the heater is connected to one of the Media analogue outputs.



This apparatus was used because it had the advantage that it was already built, operational and easily connected to Media with minimal further hardware construction. The disadvantage of the apparatus is that changes caused by the students' control tasks are not visually apparent - a system to control the level of water in a tank, for example, would have been far better in this respect.

The students' project involved the development of software to control the temperature of the air emerging from the column, using a simple proportional- plus-integral control algorithm. This

software had to be divided into three concurrently executing tasks:

1. The Operator Control Panel (OCP) task.

   This task allows the operator at any time to change the setpoint temperature or display on the screen information about the column (time, setpoint, temperature and power output).

2. The control task.

   This task performs the actual control. It reads in from the analogue input the value representing the temperature of the air and then converts it to temperature in degrees Centigrade using a calibration function to compensate for the non-linearity of the thermistor. The difference between this actual temperature and the desired (setpoint) temperature as set up by the OCP task is computed. This temperature error value is fed into the proportional-plus-integral control algorithm which calculates the amount of power which the heater must generate. The power is then written to the analogue output.

3. The logging task.

   This task logs all relevant information about the column (time, setpoint, temperature and power) to a disc file at regular intervals. This file may be examined later or printed out.

Inter-task communication is done using the WRCOMMINT and RDCOMMINT procedures from the user interface MEDUSER - for example, one of the bits in the common integer may represent an instruction from the OCP task to stop logging.

10.1 Simulation.

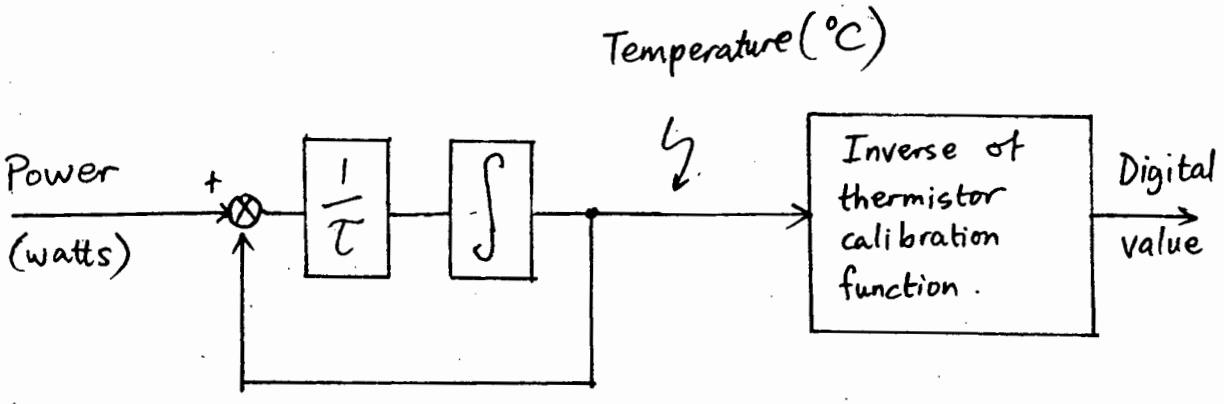
The existence of only one set of apparatus meant that only one student could use the real system at a time. Therefore, simulation software was written which uses the simulation package to simulate the operation of four other sets of apparatus (four sets because there are four simulation analogue outputs in the data-base). Thus, the system could be used by up to five students simultaneously.

The algorithm simulating the column operation is a simple first-order one:

$$\frac{\text{Temp out}}{\text{Power in}} = H(s) = \frac{A}{1+s\tau}$$

where it was assumed that the time constant τ may be different during heating and cooling of the column. The values of A, $\tau_{heating}$ and $\tau_{cooling}$ were determined empirically by taking readings of temperature against time for known values of heater power.

Thus the simulation of the column could be done using the following analogue block diagram:



The simulation task reads the power written by the user to the simulation analogue output and uses it as input to the simulation block above. The temperature, before being written to the simulation analogue input for the user program to read, must first be converted to the same digital value that would be produced by the thermistor in the real system. This is done using a look-up table to representing the inverse of the thermistor's calibration function.

The simulation of all four columns is done in parallel, and data is read from and written to the simulation portion of MEDCOM every second.

10.2 Results

The use of the computer by up to five students simultaneously for program editing, compilation and running, together with the simultaneous execution of the Media update task and the simulation task caused the computer to be very heavily loaded at times,

slowing it down considerably. This proved to be the major problem encountered. The only apparent solution to this is to reduce the number of students or to obtain a faster computer, the latter solution being a more practicable one.

No problems were experienced with the securing and releasing mechanism for MEDCOM because all students used the special Media abort task ABM to abort their tasks. As a result, it was never necessary to "unhang" MEDCOM using the data-base manager task UNHANG.

However, the students were less well disciplined regarding detaching from outputs after use. It was occasionally necessary to free outputs by detaching users who had left without detaching. In this regard, MEDRMD proved itself to be an invaluable management tool.

This student project involved heavy use of all software and hardware components of the system over a period of some seven weeks. During this time, the system was also being used for a final-year research project by a chemical engineering student, and it is therefore felt that the fact that the system ran successfully during the entire period of the projects demonstrates the viability and practicality of the multi-user process interface developed in this thesis.

REFERENCES

1. "The MEDIA Family: Technical Background", manufacturer's publication no A1ØØ7-3Ø and A1ØØ7-1, Fisher Controls Limited, New Parks, Leicester, England.

2. Parry, J.N. (Ed.): Documentation received with the GEC Media system supplied to UCT (not titled), GEC Process Control Company, Johannesburg,1982.

3. Bleach, I.C.: "Stand-alone process control computer system", undergraduate thesis number 2Ø, 1981, Department of Electrical Engineering, University of Cape Town.

4. "SMT operating system for PDP-11", RTL/2 Reference 122, SPL International.

5. "RSX-11M version 4.Ø real-time operating system", Digital Equipment Corporation software product description SPD 14.35.18, April 1982.

6. Barnes, J.G.P.: "RTL/2 Design and Philosophy", Heyden, London, 1976.

7. DEC RSX-11 RTL/2 user manual, SPL International, London, March 198Ø. Chapter 5.

8. Dehning, R.W. "Using RTL/2 under RSX-11M - a view on support environments", paper presented at the international RTL/2 users group (RUG), Brighton, October 1982.

9. Dehning, R.W.: "Using RTL/2 under the DEC RSX-11M executive (in an environment similar to MTS)", AECI Ltd., 5 Nov 1981.

1Ø. "Resource Monitoring Display (RMD)", chapter 6 of RSX-11M Version 4.Ø System Management Guide, Digital Equipment Corporation, March 1982.

11. [2] above, sections 2 and 3.

12. [2] above, section 7.

13. "Microcomputer processor handbook", Digital Equipment Corporation, 198Ø. Pages 78 to 87.

14. RSX-11M Version 4.Ø Executive Reference Manual, Digital Equipment Corporation, November 1981. Pages 2-6 to 2-11.

15. [14] above, pages 5-112 to 5-117.

16. National Semiconductor interface data book 198Ø, pages 1-88 to 1-97.

17. [16] above, page 10-13.

18. [9] above, section 6.2.

19. Conte, S.D. and de Boor, C.: "Elementary numerical analysis, an algorithmic approach", second edition, McGraw Hill, 1972, page 351.

20. "4010 and 4010-1 maintenance manual", manual number 070-1183-00, Tektronix Inc., Beaverton, Oregon.

21. [14] above, pages 5-8 to 5-9.

22. RSX-11M System Generation and Installation Guide, Digital Equipment Corporation, November 1981, Chapter 3.

23. [9] above, section 6.6.2.

24. [14] above, pages 5-14 to 5-15.

APPENDIX A

MEDIA FRONT PANEL

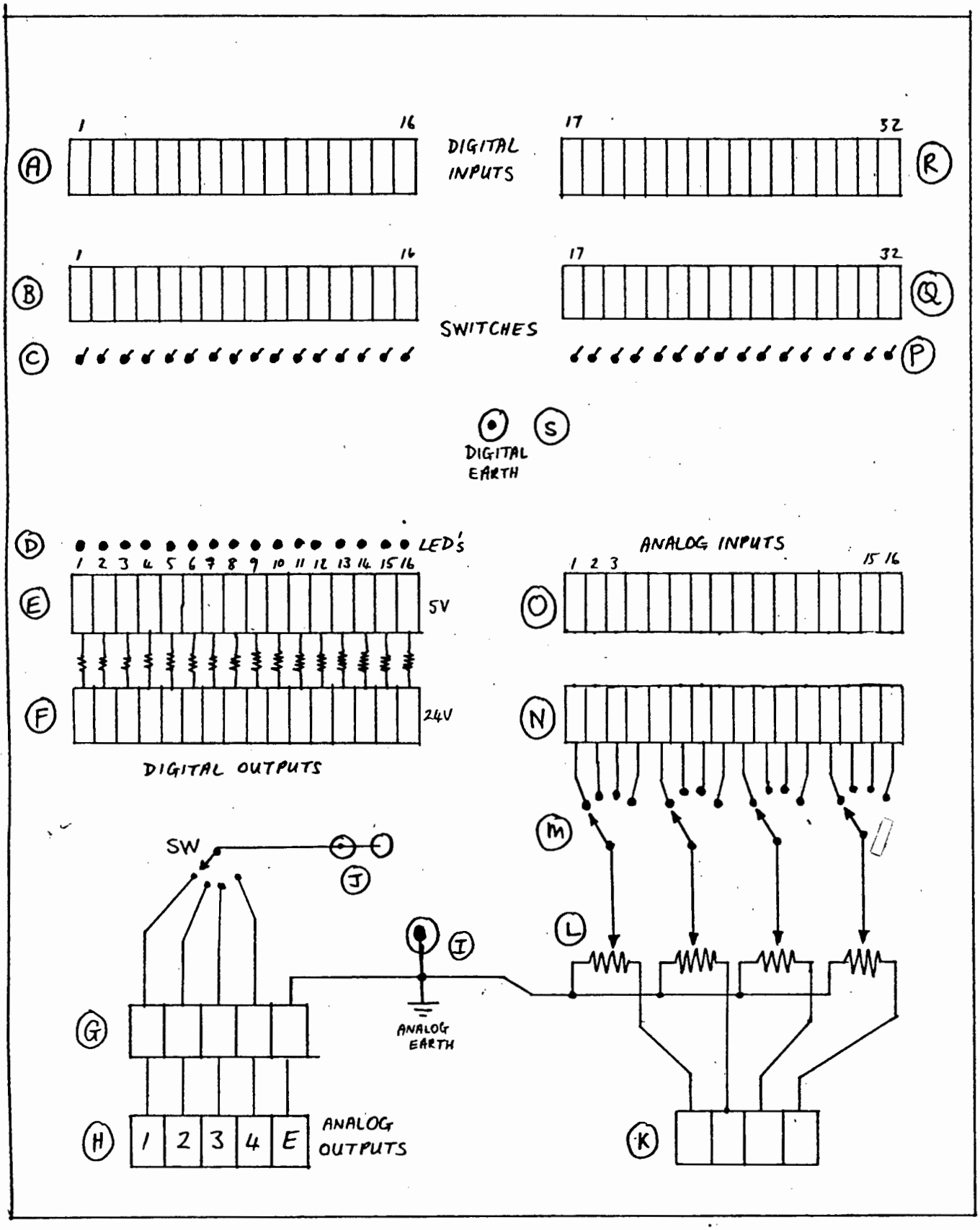


Figure A-1 : Media Front Panel.

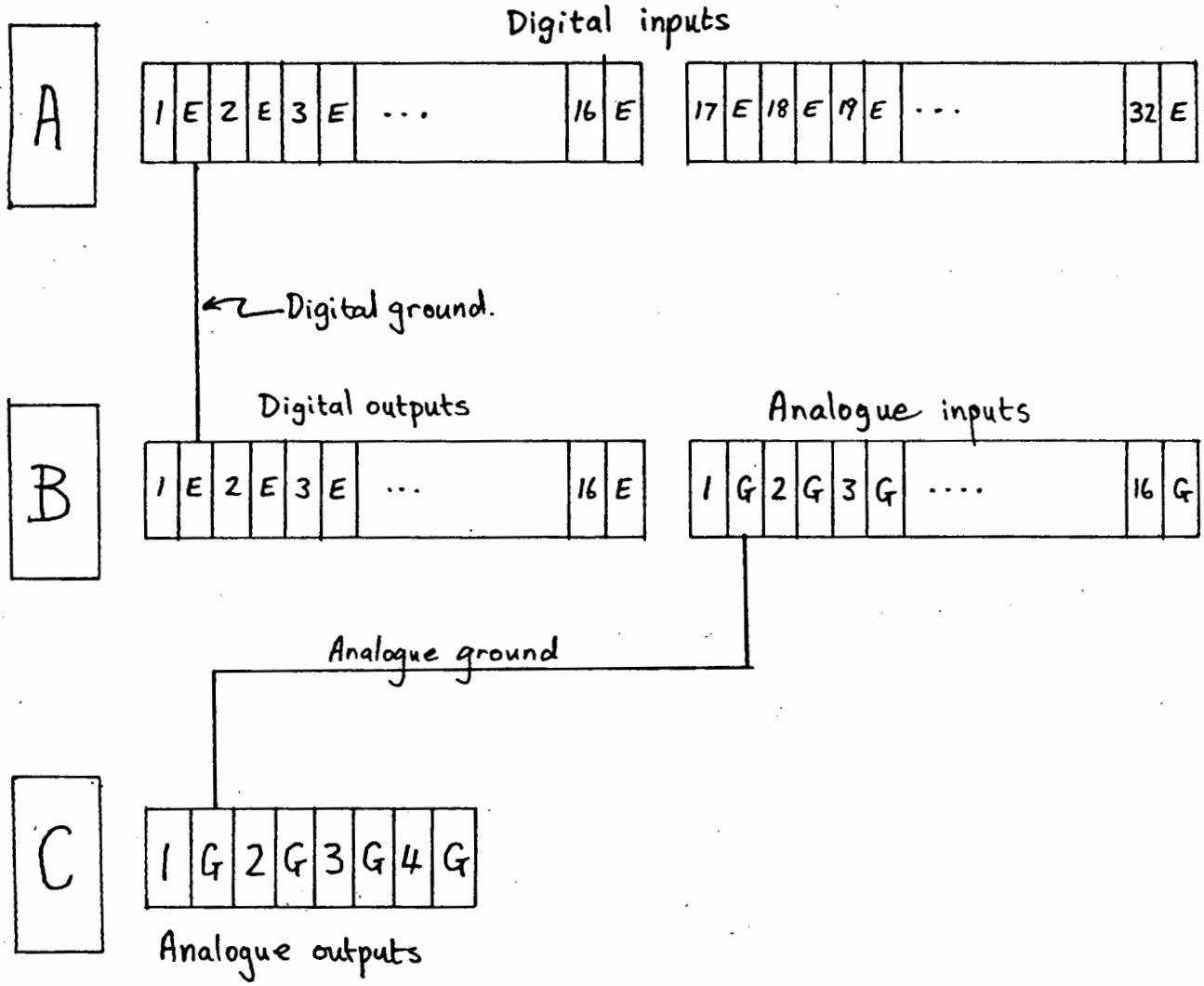


Figure A-2 : Media connection rails

The layout for the front panel of the Media system is shown in Figure A-1. This panel was designed in the Electrical Engineering digital laboratory and brings the analogue and digital output and input lines from the connection rails at the back of the Media housing (see figure A-2) to more easily-used connection blocks at the front of the housing. The Media lines are also level-shifted where this was felt desirable.

NOTE

Because of the way the Media cards have been designed, the analogue and digital Media cards use different earth reference levels. In fact,

Digital ground = Analogue ground - 15 volts.

Care must therefore be taken to ensure that one is referencing signals to the correct ground.

A.1 Digital Outputs

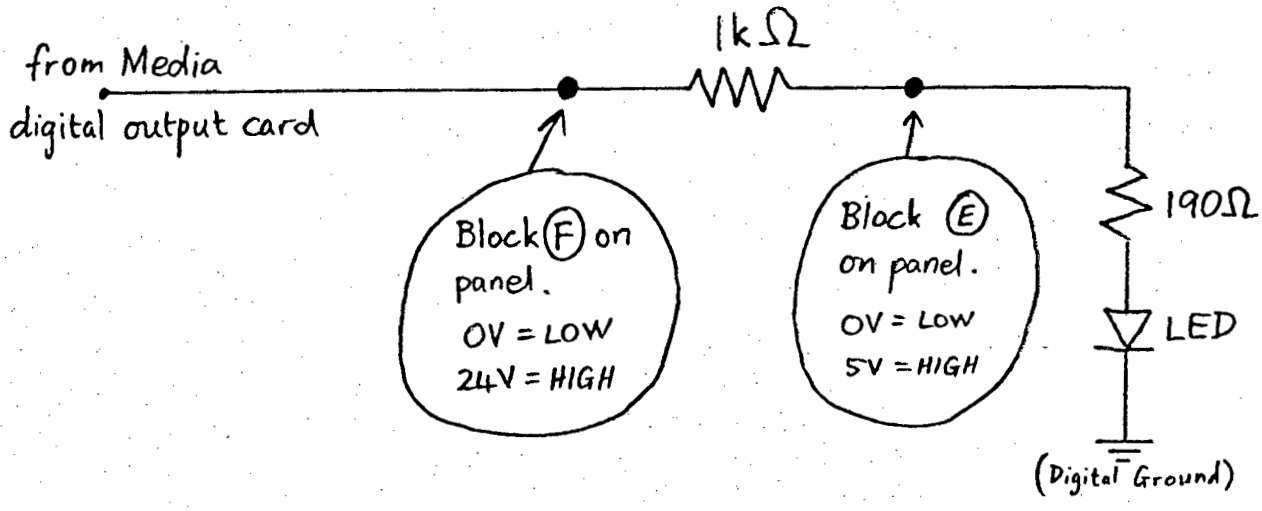


Figure A-3.

Each of the sixteen digital output lines is wired as in figure A-3. Thus all digital outputs are always displayed on the LEDs at (D) on the panel, and two levels are available - one at 24V and one TTL-compatible.

A.2 Digital Inputs

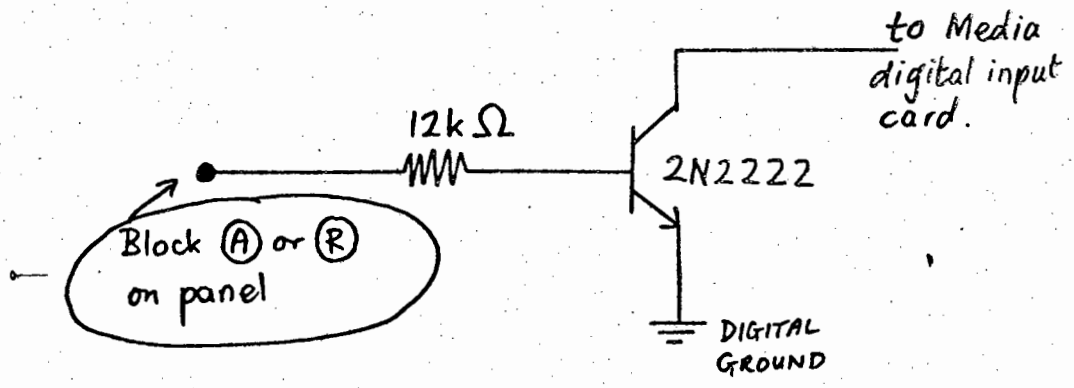


Figure A-4.

Blocks (A) and (R) on the panel are connection blocks enabling TTL-level signals to drive the digital inputs. In fact, any level

from about 3 volts to 24 volts will drive the inputs. The wiring of the inputs is shown in figure A-4. This circuitry was necessary because the inputs on the Media cards are intended to be operated by switches, not voltages.

Note that the circuit above is an inverting circuit : the inversion thus caused is undone in the software (procedure RDDIGINP in MEDUPDAT).

A.3 Switches

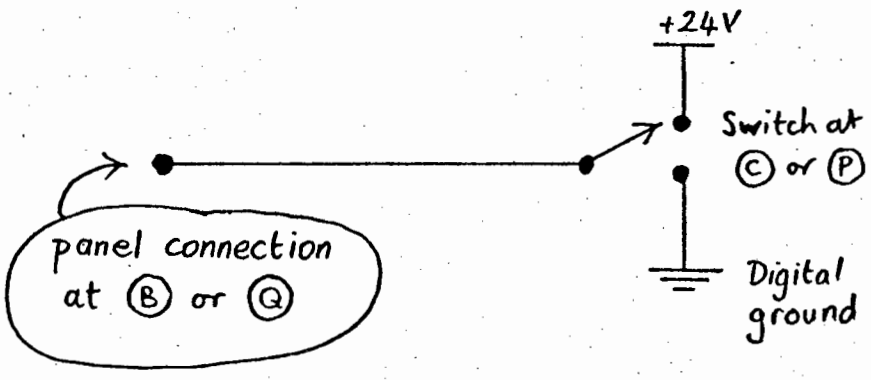


Figure A-5.

32 switches, wired as in figure A-5, are provided. These can be connected directly to the 32 digital inputs, as sense switches or for testing purposes. Switch up corresponds to logical 1 (or +24 volts) and switch down corresponds to logical 0 (or 0 volts).

A.4 Analogue Outputs

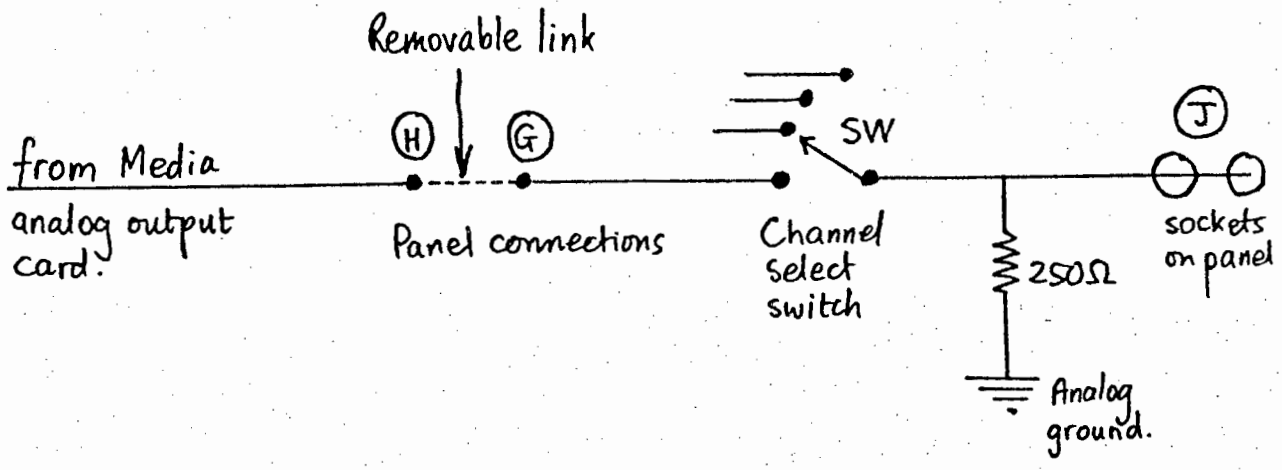


Figure A-6.

Each of the four analogue outputs is brought to a connection on block (H) on the panel. In addition, analogue earth is the rightmost connection on block (H).

-4 mA to -20 mA outputs are available from block (G) if the wire links between blocks (G) and (H) in the appropriate places are removed. Otherwise, the 4-way switch can be used to select one of the analogue outputs, which is then available as a -1 to -5 volt output at (J). The current-to-voltage conversion is done by a 250 ohm resistor, as shown in figure A-6.

A.5 Analogue Inputs

The 16 analogue input lines to the multiplexer card are connected to block (G) of the panel. -1V corresponds to an ADC output of zero, and -5V to an ADC output of HEX 3FF.

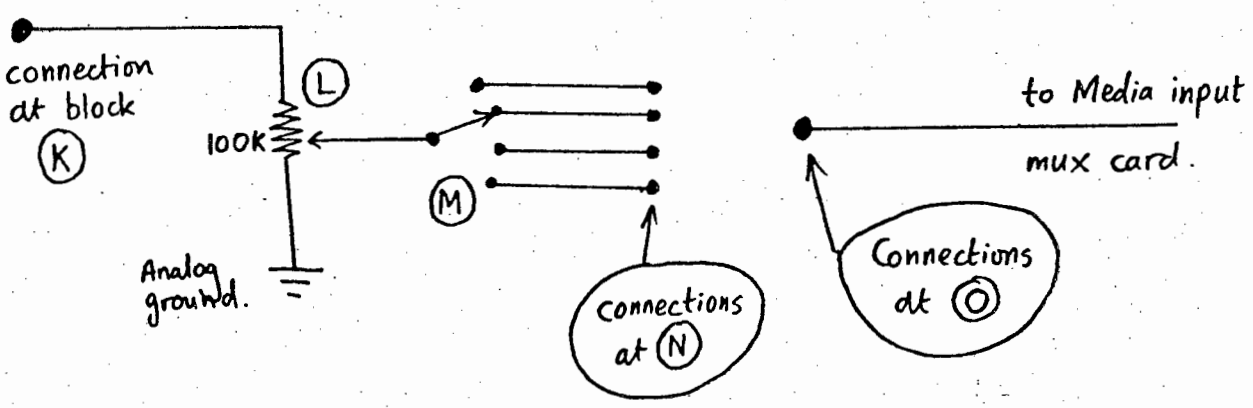


Figure A-7.

In addition, analogue signals can be connected to block (K), pass via attenuators (L) and switches (M) to block (N), and hence to block (O) via wire links. This is shown in figure A-7.

A.6 Connections to the Media connection blocks

The connections to the connection blocks at the back of Media are shown in figure A-2. The input/output lines from the Media cards are brought directly to the terminal points shown.

Digital ground is obtained by connecting the earth lines (from the Media cards) of the digital inputs and outputs together. Analogue ground is produced similarly.

APPENDIX B

ADDRESSING THE MEDIA CARDS

This section describes how the systems programmer interacts with the Media cards using the LSI-11.

B.1 Converting between Media and LSI-11 addresses.

A Media address is a ten-bit address of a card on the Media highway. When the PDP-11 to Media interface cards are present in the minibin, successive Media addresses are available to the LSI-11 as successive memory-mapped words in the I/O space of the LSI, starting from address octal 764000.

The formula for converting Media addresses to LSI-11 addresses is

LSI address = 764000 + (2 * Media address)

and to convert LSI addresses to Media addresses,

Media address = (LSI address - 764000) / 2.

B.2 Addresses of the Media cards

Media Card	Media address	LSI address
16-way dig. inp. card 1	Ø	764ØØØ
16-way dig. inp. card 2	1	764ØØ2
16-way digital output card	4 (octal)	764Ø1Ø
16-way analogue mux card	1Ø	764Ø2Ø
A/D converter card	14	764Ø22
4-way D/A converter card	14	764Ø3Ø

Table B-2: Addresses of the Media cards

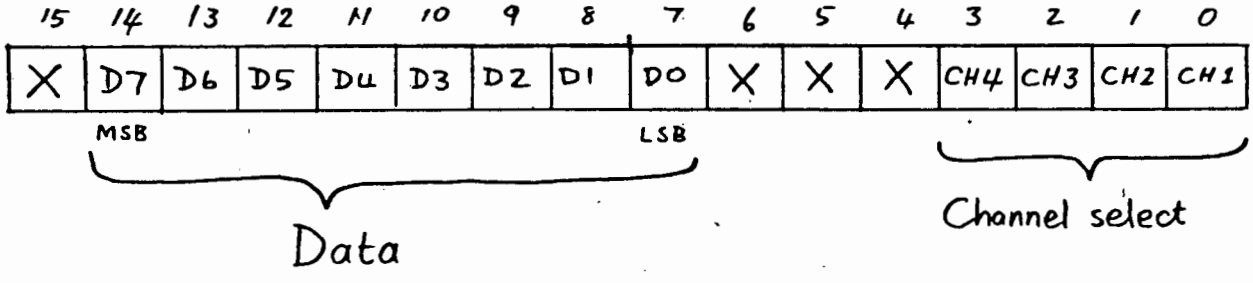
The Media and LSI addresses of the Media cards are given in Table B-1. In addition, the Media status word is available at LSI address 767776 (Media address 1777) and can be interpreted as :

000000 = last transaction successful
010000 = device missing error

B.3 Accessing the cards

B.3.1 The digital-to-analogue converter card

This card is a four-channel 8-bit digital-to-analogue converter. The word at Media address OCT 14 is organised as follows :

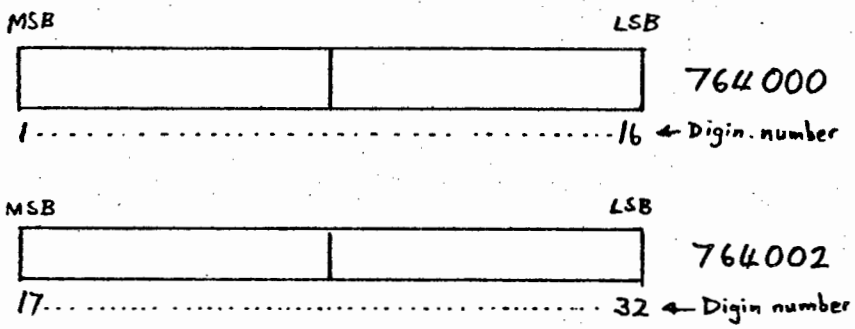


Any of the four channel-select bits can be used; so that the same data can be written to more than one output simultaneously if required. However, Micro-Media does not use this feature.

Each analogue output is a current signal in the range -4mA (for digital 0) to -20mA (for digital HEX FF).

B.3.2 The digital input cards

These are read as two consecutive words at Media addresses 0 and 1 for digital inputs 1-16 and 17-32 respectively.



Digital input number 1 the the most significant bit of the first word, and input number 32 is the least significant bit of the second word, as shown in the diagram.

The digital input cards require switch inputs, but have been wired to the front panel in such a way that either switches or TTL logic levels can be used.

B.3.3 The digital output card

These may be written to at LSI address 764010. Digital output 1 is the most significant bit and output 16 is the least significant bit.

The Media card provides 24V = logical 1 and 0V = logical 0. The outputs are also available on the front panel as TTL compatible levels.

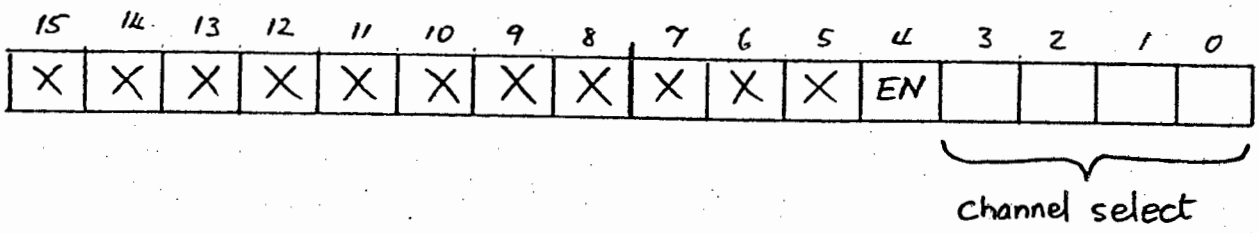
B.3.4 The analogue inputs

These 16 inputs are organised as a 16-channel multiplexer card at LSI address 764020 and an ADC card at address 764022.

To read an analogue input, the following procedure is followed :

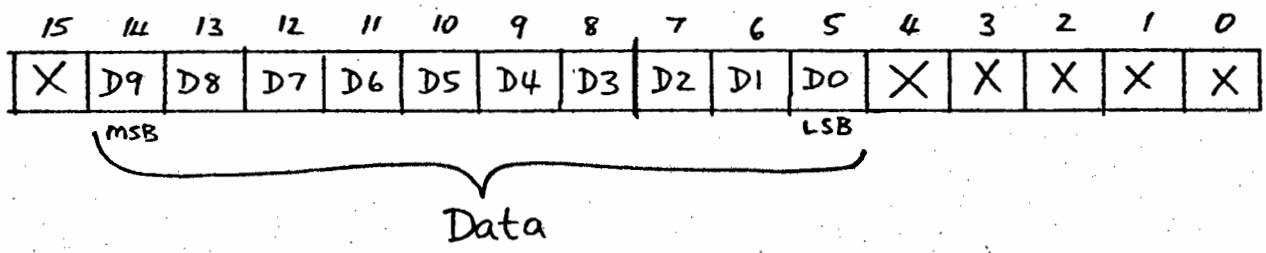
1. Select the required input by writing to the multiplexer.
2. Wait for the multiplexer to settle (this takes about 50 milliseconds).
3. Read the digital value from the ADC card.

B.3.4.1 The multiplexer card - The multiplexer card word at LSI address 764020 is organised thus:



To select analogue input n, a word must be written to the multiplexer with the enable bit (bit 4) set, and bits 0 to 3 must contain the number n-1. For example, to select channel 4, one must write OCT 000023 to the multiplexer.

B.3.4.2 The ADC card - The ADC card word at LSI address 764022 is organised as follows :



The data is read as the 10 bits from bit 5 to 14 (inclusive) of the word. -1V corresponds to digital zero, and -5V to digital HEX 7FE0.

APPENDIX C

DETAILED DESCRIPTION OF MEDIA LINK FRAME STRUCTURE

This appendix describes in detail the format of the Media data link frames introduced in chapter 2.

C.1 Symbols used

The descriptions below use the following symbols:

1. TA1 to TA4 are the 4-bit terminal address
2. P is the parity bit
3. A0 to A9 is the 10-bit Media address
4. D0 to D15 is the 16-bit data field
5. BCC is the block check character
6. X represents a don't-care bit

C.2 Single read (Codes 1,2,8)

The host-to-Media frame is as follows :

MSB LSB

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	A3	A2	A1	A0
P	0	A9	A8	A7	A6	A5	A4
P	1	BCC					

The Media-to-host reply is as follows :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	STATUS			
P	0	D5	D4	D3	D2	D1	D0
P	0	X	D10	D9	D8	D7	D6
P	0	X	D15	D14	D13	D12	D11
P	1	BCC					

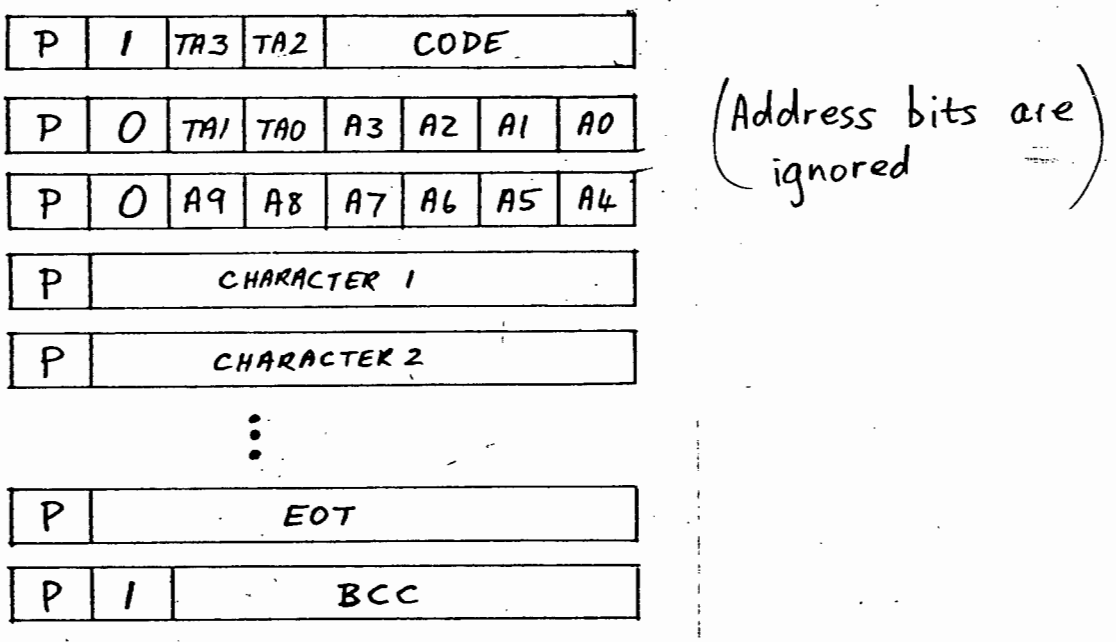
Thus the returned data is encoded into bytes 3 to 5 of the reply frame.

C.3 Printer output (Code 0)

This command causes the supplied text (CHAR1 to CHARn) to be printed at the terminal local to the Media. This is not implemented in the LSI-11 Media system as there is no local

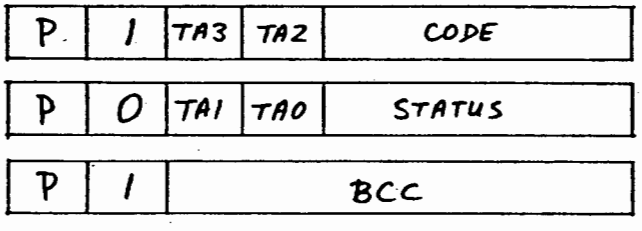
printer.

The host-to-Media frame is :



where EOT = control-D = binary 0000100.

The Media-to-host reply is :



C.4 Block read (Codes 3 and 4)

The Media or list address of the start address of the block is encoded into bytes 2 and 3 of the host-to-Media frame. The number of items required is put into byte 4. The frame is :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	A3	A2	A1	A0
P	0	A9	A8	A7	A6	A5	A4
P	0	NUMBER OF ITEMS TO READ					
P	1	BCC					

The Media-to-host reply is :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	STATUS			
P	0	D5	D4	D3	D2	D1	D0
P	0	X	D10	D9	D8	D7	D6
P	0	X	D15	D14	D13	D12	D11
P	0	D5	D4	D3	D2	D1	D0
P	0	X	D10	D9	D8	D7	D6
P	1	X	D15	D14	D13	D12	D11
P	1	BCC					

First word

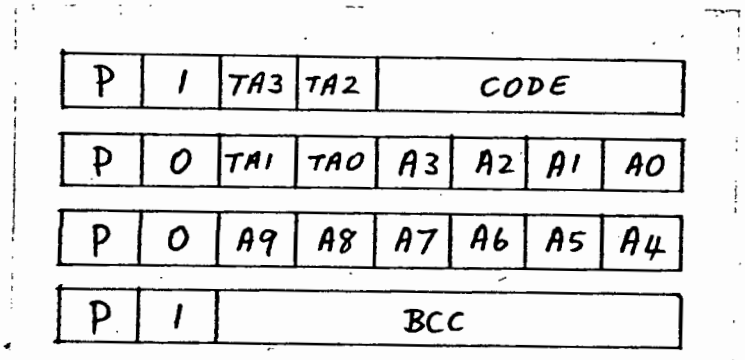
Second word

The k'th data item read is encoded into bytes 3k to 3k + 2 of the reply frame, as shown.

C.5 Keyboard input (Code 9)

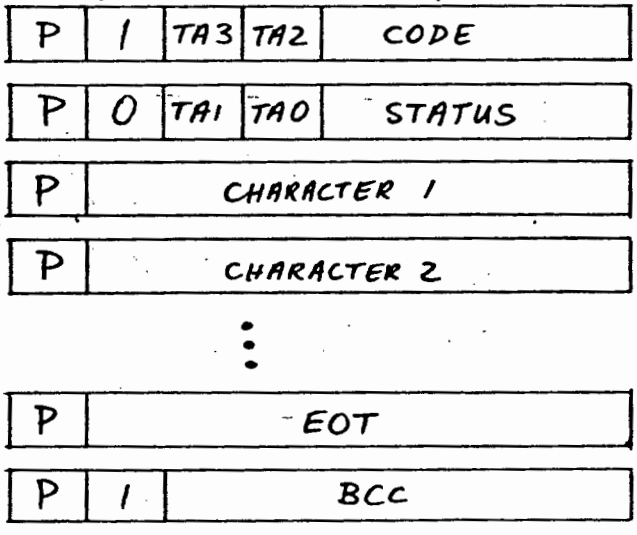
This is used for the host to read in the keyboard buffer of the terminal local to the Media. This is not implemented in the LSI-11 Media system.

The host-to-Media frame is :



(Address bits are ignored)

The Media-to-host reply is :



C.6 Single write (Codes C, D and E)

The write is implemented as a two-stage process. First the host issues a Media/list write frame (code C or D), as follows :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	A3	A2	A1	A0
P	0	A9	A8	A7	A6	A5	A4
P	0	D5	D4	D3	D2	D1	D0
P	0	X	D10	D9	D8	D7	D6
P	0	X	D15	D14	D13	D12	D11
P	1	BCC					

Media then does nothing except to echo this frame, in its entirety, back to the host for checking. If the host is satisfied with this reply, it sends a confirmatory 'Go' frame (code E), as follows :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	A3	A2	A1	A0
P	0	A9	A8	A7	A6	A5	A4
P	1	BCC					

←

The Media then performs the write and sends the following reply :

P	1	TA3	TA2	CODE
P	0	TA1	TA0	STATUS
P	1	BCC		

C.7 Read digital change words (Code 7)

The host-to-Media frame is :

P	1	TA3	TA2	CODE			
P	0	TA1	TA0	A3	A2	A1	A0
P	0	A9	A8	A7	A6	A5	A4
P	1	BCC					

where the address field is ignored by Media.

The Media-to-host reply is :

P	1	TA3	TA2	CODE				
P	0	TA1	TA0	STATUS				
P	0	D5	D4	D3	D2	D1	D0	First change word.
P	0	X	D10	D9	D8	D7	D6	
P	0	X	D15	D14	D13	D12	D11	
P	0	D5	D4	D3	D2	D1	D0	Second change word.
P	0	X	D10	D9	D8	D7	D6	
P	0	X	D15	D14	D13	D12	D11	
P	1	BCC						

C.8 Media frame lengths

Table C-1 shows the lengths of all the Media frames in bytes.

Code	Meaning	Length (bytes) Command	Reply
Ø	Printer output	variable	3
1	Read Media address	4	6
2	Read list address	4	6
3	Read block Media addresses	5	3(n+1)
4	Read block list addresses	5	3(n+1)
5	Spare	-	-
6	Spare	-	-
7	Read digital change words	4	9
8	Read printer status word	4	6
9	Read keyboard input	4	variable
A	Spare	-	-
B	Spare	-	-
C	Write to Media address	7	7
D	Write to list address	7	7
E	Go (2nd stage of write)	4	3

Table C-1: Media link frame lengths

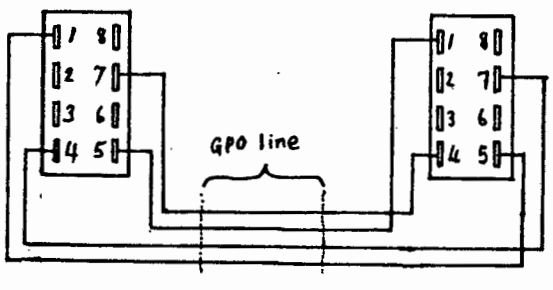
APPENDIX D

SERIAL LINK HARDWARE CIRCUIT DIAGRAMS

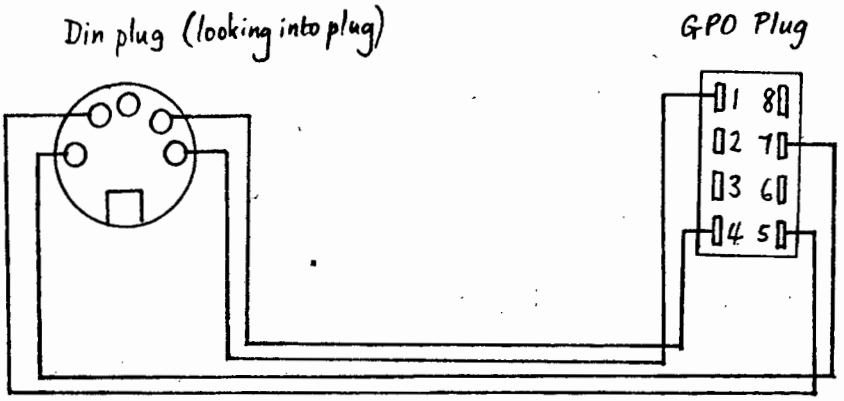
D.1 The serial lines

The wires of each 4-wire serial link between the Electrical and Chemical Engineering buildings are connected as follows:

Elec Eng wall socket pin number	Chem Eng wall socket pin number
1	5
4	7
5	1
7	4

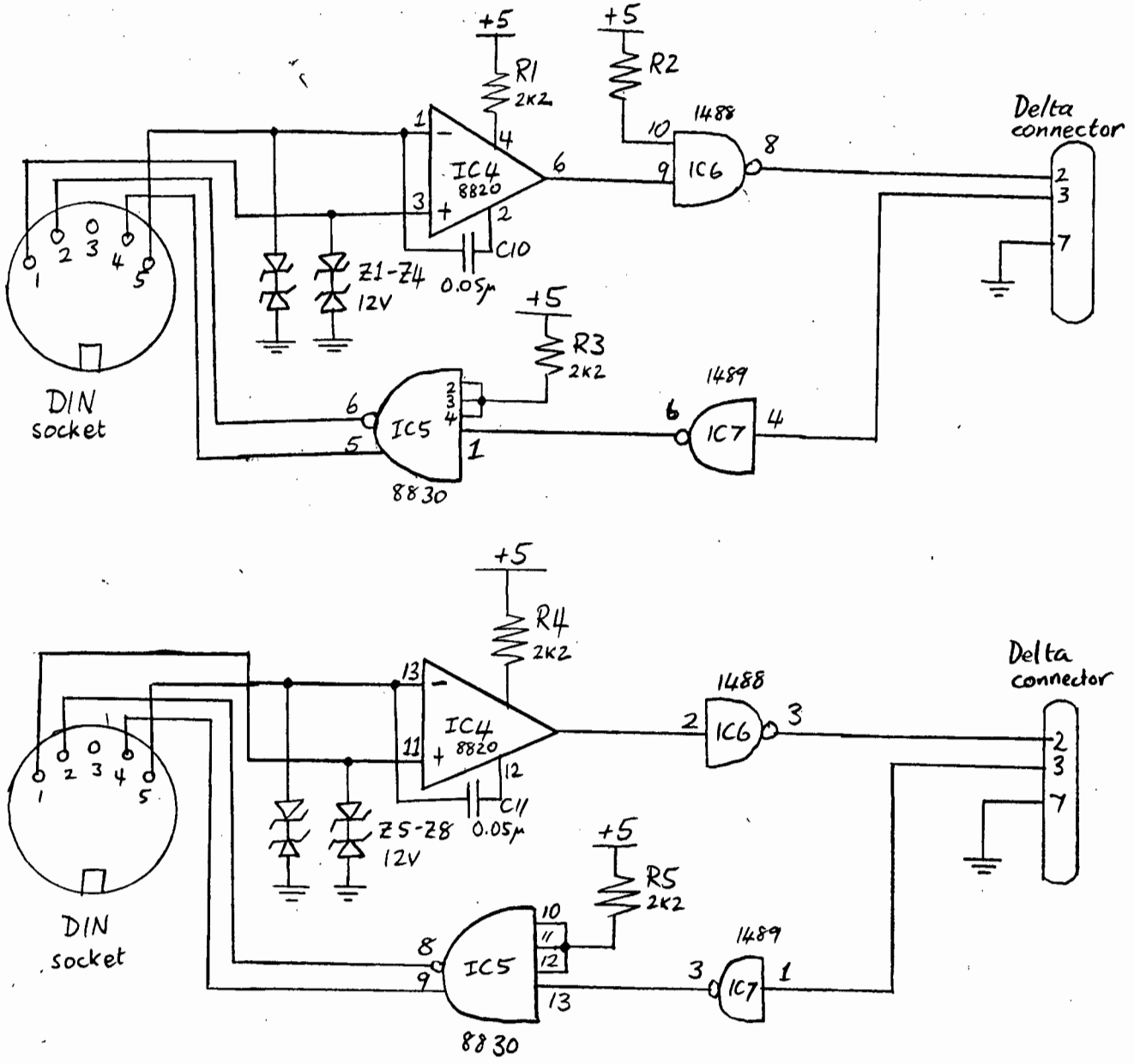


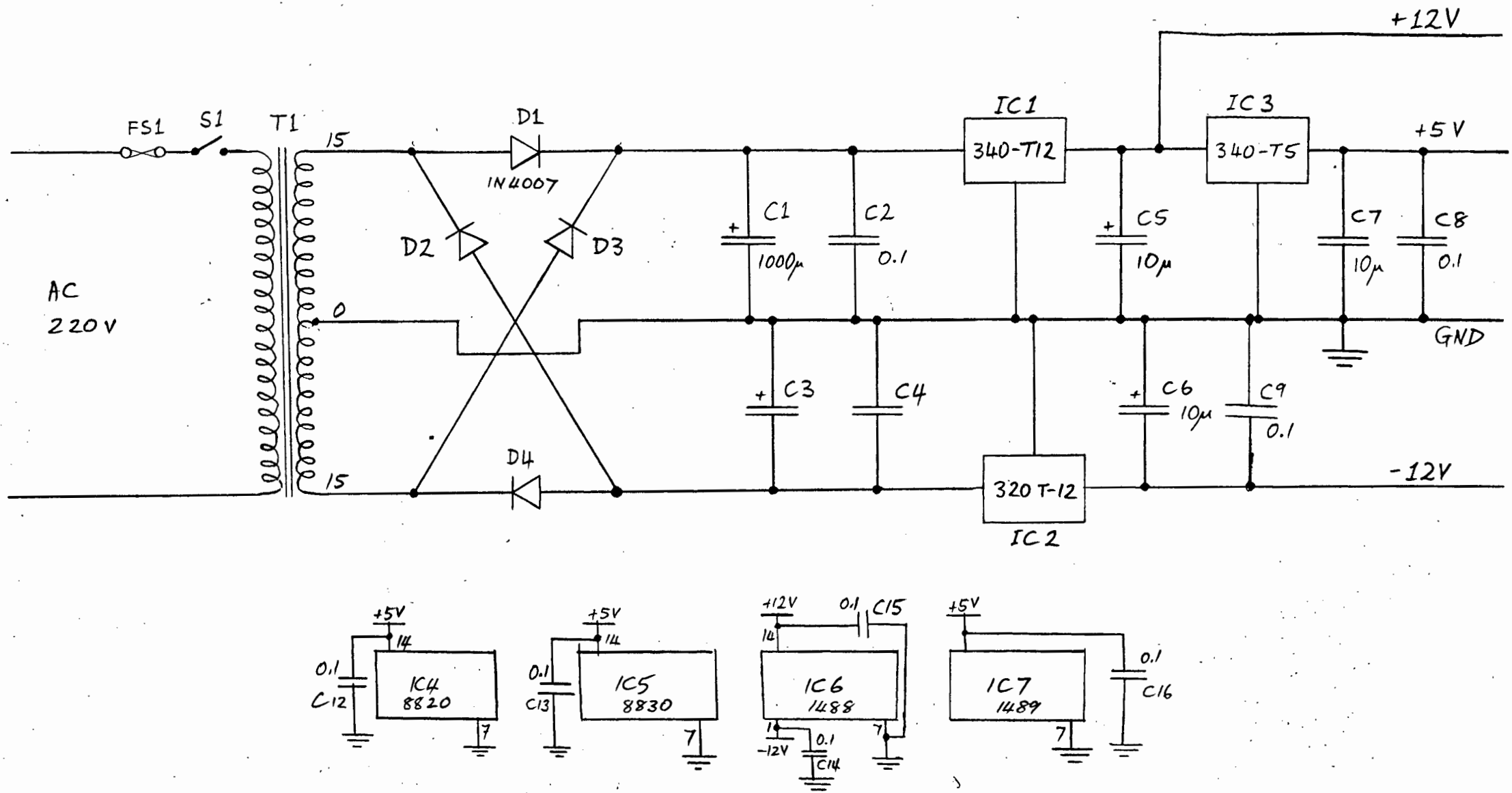
The lead connecting the wall GPO socket to the DIN socket on the interface unit is wired thus:



D.2 The link driver circuitry

Full circuit diagrams and parts list follow.


+5
R1
2K2
+5
R2
1488
IC4
8820
IC6
Delta
connector
DIN
socket
Z1-Z4
12V
0.05μ
C10
+5
R3
2K2
1489
IC5
IC7
8830
+5
R4
2K2
1488
IC4
8820
IC6
Delta
connector
DIN
socket
Z5-Z8
12V
0.05μ
C11
+5
R5
2K2
1489
IC5
IC7
8830


+12V
FS1
S1
T1
15
D1
IN4007
D2
D3
AC
220V
0
15
D4
C1
1000µ
C2
0.1
IC1
340-T12
C5
10µ
IC3
340-T5
+5V
C7
10µ
C8
0.1
GND
C3
C4
C6
10µ
C9
0.1
320 T-12
IC2
-12V
+5V
14
0.1
C12
IC4
8820
7
+5V
14
0.1
C13
IC5
8830
7
+12V
0.1
C15
14
IC6
1488
1
-12V
0.1
C14
7
+5V
IC7
1489
7
0.1
C16

SERIAL LINK HARDWARE CIRCUIT DIAGRAMS

PARTS LIST

IC1 : LM340-T12 voltage regulator
IC2 : LM320-T12 voltage regulator
IC3 : LM340-T5 voltage regulator
IC4 : DS8820 dual differential line receiver
IC5 : DS8830 dual differential line driver
IC6 : DS1488 quad RS-232 line driver
IC7 : DS1489 quad RS-232 line receiver

D1-D4 : 1N4007 diode
Z1-Z8 : 12 volt zener

R1-R5 : 2K2 1/4 watt resistor

C1,C3 : 1000 uF 25V electrolytic
C2,C4,C8,C9,C12-C16 : 0.1uF 25V ceramic disc
C10,C11 : 0.05 uF ceramic disc

T1 : Transformer, 220 volt to 15-0-15 volt 1 amp

FS1 : 500 mA fuse

S1 : SPST toggle switch 250 volts

Two 25-pin panel-mounting male delta connectors
Two 5-pin panel-mounting DIN sockets

```
ENT PROC GETMED (REF MEDCARD MSTAT) INT;
```

which puts the Media status word into MSTAT.MEDDAT, updates MSTAT.SCANTIME and returns an integer indicating error status (as for the other procedures in MEDLNK).

APPENDIX F

THE ABM AND ABOM ABORTING TASKS

The two tasks ABM and ABOM are provided for use instead of the RSX-11M abort command ABO, and are intended for aborting applications tasks which access MEDCOM. ABO cannot be used for aborting such tasks because it does not ensure that the task releases MEDCOM before it exits.

ABM is a non-privileged task which should be installed as ...ABM and uses the same command-line format as ABO. ABM calls GETMCR() to get the command-line typed in, and from this, gets the name of the task to be aborted. It converts the task-name into radix-50 task name format. If the user did not specify a task name, then the usual default name is used (i.e. name "TT1" if invoked from terminal TT1:, etc.)

Once the task name is obtained, ABM secures MEDCOM using SECMEDCOM(), the task is aborted using the RSX-11M ABRT$S directive [21], and then MEDCOM is released. This ensures that the task being aborted will not be secured to MEDCOM on exit.

Two checks are necessary before the ABRT$S can be issued:

1. The task to be aborted must not be "MCR...". If a user

ABOM because privileged tasks cannot be installed from non-privileged terminals.

The above method of having two abort-MEDIA tasks is intended only as a temporary measure until a way is found around the difficulty mentioned above. There definitely is a solution, since the RSX-11M TAL command, given a task name, prints out information which includes the terminal from which the task was run.

APPENDIX G

THE SECURE/RELEASE MODULE GSECREL

G.1 Procedures provided in GSECREL

In addition to the procedures relating to the securing and releasing of facilities, three other procedures (the final three in the list below) are included in this module because they are often used by the same tasks that use the secure/release procedures.

1. ENT PROC GSECURE (INT FA);

   This proc secures the facility represented by event flag FA. It is similar to the MTSLIB SECURE, but uses the global event flags 33 to 56.

2. ENT PROC GRELEASE (INT FA);

   This proc releases the facility represented by the global event flag FA.

3. ENT PROC FORCEDRELEASE (INT FA);

   This is the same as RELEASE except that it releases the facility irrespective of who has secured it. It is intended for use by a system manager to free a facility that has become "hung" (as explained earlier).

4. ENT PROC SECMEDCOM ();

   Secures the data-base, using GSECURE with the event flag MEDCOMEF (= 33 at present).

5. ENT PROC RELMEDCOM ();

   Releases the data-base, using GRELEASE with the event flag MEDCOMEF.

6. ENT PROC MCOMINIT ();

   Sets the flag MEDCOMEF. This is meant to be called once only, as part of the RSX-11M system startup, and corrects for the problem mentioned in section 6.3.2 whereby the flags "wake up" in the reset state when the system is started up.

7. ENT PROC FREEMEDCOM ();

   This proc calls FORCEDRELEASE (MEDCOMEF) to free MEDCOM should it "hang". It is intended for use only by the data-base manager [300,1] and is called only by the task UNHANG.

8. ENT PROC MCOMTASK (REF R50NAME TASK);

   This proc returns the radix-50 name of the task which is currently securing MEDCOM. If no task is securing, TASK is zeroed. It is called by MEDUPDAT.

9. ENT PROC PRIVILEGED () INT;

   This proc returns 1 if the calling task was run from a privileged terminal, otherwise it returns 0. The calling task must have logical unit 1 assigned to TI:.

10. ENT PROC TASKINFO (REF TASKBUF TB);

    This is the same as the SPL interface proc RSXGTS except that it RRGELs if the GTSK$S directive fails.

G.1.1 Modifications to RSXBA2

As mentioned in chapter 6, the MTSLIB-supplied version of RSXBA2 stores the names of the tasks securing facilities. To do this, it uses a section of code equivalent to

```
MODE R50NAME ( INT R50N1, R50N2);
ENT DATA RRFACS;
    ARRAY(32) R50NAME FACS;
ENDDATA;
```

Element i of FACS contains the radix-50 name of the task securing the facility represented by event flag 64 + i.

To enable the secure/release mechanism using the global event flags (33 to 56), two extensions to RSXBA2 have been made :

1. code equivalent to

```
MAC RSXBA2=LB:[1,1]EXEMC/ML,SY:[100,6]RSXBA2
```

RTLLIB is then built using

```
TKB @RTLLIBTKB
```

where RTLLIBTKB.CMD is

```
RTLLIB/-HD,RTLLIB/-SP,RTLLIB=LB:[1,1]RTLCTL
SY:[100,6]RSXBA2,RRGEL
SY:[100,6]TTOUT,DEVIO,TWRT,IWRT,IWRTF,NLS,SPS
SY:[100,6]IREAD,TREAD,RWRTX,RWRTU,RREADU
/
STACK=0
PAR=RTLLIB:120000:14000
//
```

The details of forming the RTLLIB partition during system VMR are given in appendix H.

APPENDIX H

CONSTRUCTION OF MEDCOM AS A RESIDENT COMMON

H.1 Compilation and building of the data-base

Medcom is built using the following commands:

```
RTL MEDCOM,MEDCOM=MEDCOM
MAC MEDCOM=MEDCOM
TKB @TKBMEDCOM
```

where TKBMEDCOM.CMD consists of:

```
MEDCOM/-HD,MEDCOM/-SP,MEDCOM=MEDCOM
/
STACK=0
PAR=MEDCOM:140000:1100
//
```

The line PAR=MEDCOM:140000:1100 tells the taskbuilder that the

output file will be installed in a partition called MEDCOM, of size 1100 (octal) bytes starting at virtual address 140000 octal.

H.2 Creating partitions for MEDCOM and RTLLIB

RTLLIB, which is a library of RTL/2 procedures and contains the extended version of the RSXBA2 procedure which is needed by the global event-flag secure/release procedures, is introduced in appendix G, where it is also explained why RSXBA2 must reside inside a common area of memory.

This section deals with the procedure which must be followed when building the partitions RTLLIB and MEDCOM. This is done at VMR time during Phase two of RSX-11M system generation [22].

The file DL0:[1,54]SYSVMR.CMD must be edited to insert the lines

```
SET /MAIN=MEDCOM:*:11:COM
SET /MAIN=RTLLIB:*:147:COM
```

just before the line

```
SET /MAIN=GEN:*:*:SYS
```

to inform VMR that MEDCOM and RTLLIB will be common partitions of size 11 (octal) and 147 (octal) 64-byte blocks respectively.

VMR is then performed as follows:

```
SET /UIC=[1,54]
PIP DL2:RSX11M.SYS/NV/CO/BL:258.=DL0:RSX11M.TSK
PIP DL2:=DL0:*.STB
ASN DL2:=SY:
ASN DL2:=LB:
INS $VMR
VMR @DL0:SYSVMR
ASN DL0:=SY:
ASN DL0:=LB:
PIP =DL2:RSX11M.SYS
```

The new system is then booted using

```
BOO RSX11M
```

The files MEDCOM.TSK, MEDCOM.STB, MEDCOM.MAP, RTLLIB.TSK, RTLLIB.STB and RTLLIB.MAP must then be copied to LB:[1,1], and the startup file dl0:[1,2]STARTUP.CMD edited to include the lines

```
INS LB:[1,1]MEDCOM
INS LB:[1,1]RTLLIB
```

H.3 Linking tasks to MEDCOM

Tasks which access MEDCOM have to be taskbuilt as follows:

```
TKB @USERTKB
```

where USERTKB.CMD includes:

```
USERTASK,USERTASK/-SP=USERTASK
@LB:[1,54]COMINTFCE
/
LIBR=RTLLIB:RW
COMMON=MEDCOM:RW
//
```

It is not good programming practice to run AST service routines in an RTL/2 environment because this can compromise the system security [23]. A module called AST, has therefore been written which provides RTL/2-callable subroutines which enable the processing of ASTs in a simple fashion which does not involve the buffering up of incoming characters, and can only supply the task with the most recent unsolicited character.

I.2 Module description

The following description should be read in conjunction with the listing of the AST module.

Logical unit number 2 must be assigned to the terminal from which the asynchronous input is expected. Asynchronous input handling is initialised by attaching to logical unit number 2 using

```
QIO$S  #IO.ATA,...,<#ASTIO,#AST2>
```

where ASTIO is the address of the AST service routine ASTIOPROC() for characters other than control-C, control-S, control-O, control-Q, and control-X ; and AST2 is the address of the AST service routine AST2PROC() for input control-C characters. The control-S, control-Q, control-O, and control-X characters do not cause ASTs, but instead have the usual effect in controlling output to the terminal.

Processing of ASTs is terminated by issuing to logical unit number 2 a

```
QIO$S  #IO.DET,...
```

to detach from the terminal.

A local data brick contains the following two integers :

1. ASTFLAG. This takes one of three values :

   CTLC : Indicates that an unsolicited control-C character was input since the last input character dealt with by the user.

   AST : Indicates that an unsolicited character other than control-C, control-S, control-Q, control-O or control-X was input since the last character dealt with.

   NOTYET : Indicates that no unsolicited characters have been input since the last input character dealt with.

2.

   CHAR : An integer containing the unsolicited input character.

The AST service routines ASTIOPROC and AST2PROC should never be called explicitly. A call to RRGEL aborts any attempt to do so.

ASTIOPROC services non-control-C ASTs. It moves the value #AST to ASTFLAG to indicate the type of AST that has occurred, pops the character off the stack and stores it in CHAR. It then sets event flag ASTEF (event flag 4 at present), declares a significant event and exits via the RSX-11M AST-exit directive ASTX$S [24]. Failure of the ASTX$S would mean a serious system fault and is dealt with by calling RRGEL.

AST2PROC, the control-C AST service routine, is the same as ASTPROC except that it sets ASTFLAG to #CTLC and sets event flag AST2EF (event flag 3 at present).

ASTFLAG is reset to NOTYET whenever a character is actually dealt

with by any of the procedures WAITAST, WAITCTLC, WAITNCTLC, WTCCCZ, ASTYET, CTLCYET and NCTLCYET. For example, ASTFLAG is reset by WAITCTLC if a control-C occurred, otherwise not.

I.3 Procedures in the module

The user may call the following ENT procedures :

1. ENT PROC STARTAST();

   This procedure initialises for AST processing by issuing a QIO$S #IO.ATA directive to attach to logical unit number 2.

2. ENT PROC WAITAST(INT NTIX) INT;

   Puts the calling task into a wait state until an AST of either type occurs. A timeout of NTIX ticks is imposed. The integer returned is 1 if an AST occurred and 0 if the wait timed out.

3. ENT PROC WAITCTLC (INT NTIX) INT;

   As for WAITAST except that the wait is for ASTs of the control-C type only.

4. ENT PROC WAITNCTLC (INT NTIX) INT;

   As for WAITAST except that the wait is for non-control-C ASTs only.

5. ENT PROC WAITCCCZ (INT NTIX) INT;

   This procedure waits until control-C or control-Z is

typed, timing out after NTIX ticks. The returned integer is 1 if control-C or control-Z was typed, zero if timeout occurred.

6. ENT PROC ASTYET () INT;

   If an AST of either type has occurred since the last character dealt with, 1 is returned; otherwise zero is returned.

7. ENT PROC CTLCYET () INT;

   Returns one if a control-C AST has occurred since the last character dealt with; otherwise returns zero.

8. ENT PROC NCTLCYET () INT;

   Returns one if a non-control-C AST has occurred since the last character dealt with; otherwise returns zero.

9. ENT PROC ASTCHAR () BYTE;

   This procedure returns the most recently received input byte. A call to ASTCHAR would be preceded by a call to one of the routines which indicates whether an AST has occurred. For example,

```
IF WAITAST(50) = 1 THEN
   INPUTBYTE := ASTCHAR ();
END;
```

   would wait for up to a second for a character, and if one was typed, put it into the variable INPUTBYTE. This technique is used in MEDRMD to allow it to respond to the pressing of any key.

10. ENT PROC STOPAST ();

This procedure issues a QIOW$S #IO.DET to detach from logical unit number 2. The effect is to stop processing of ASTs and to return the terminal to its usual state.

APPENDIX J

SCREEN CURSOR ADDRESSING ROUTINES

The module SCREEN provides easy-to-use subroutines for screen cursor position control for a VT52 terminal. It is used by several of the software modules such as MEDRMD and ATTACH, and may also be used by applications tasks to present neatly-formatted data on the screen.

The RTL/2 byte-output procedure variable OUT should ideally be assigned to HSOUT, because binary escape-sequences are output to the screen. However, in practice no problems were encountered using TTOUT.

The following cursor-routines are in the module :

1. ENT PROC GOTOLC (INT LINE, COL);

   This procedure positions the cursor on line LINE and column COL of the screen. LINE must be in the range 1..24 and COL in the range 1..80, or ERP is called.

2. ENT PROC HOME ();

   This procedure positions the cursor at the top of the

screen, by outputting "<esc> H". It has the same effect as GOTOLC (1,1).

3. ENT PROC CLEOS ();

   Clears the screen from the current cursor position to the end of the screen, by outputting "<esc> J".

4. ENT PROC CLSCREEN ();

   Clears the entire screen by calling HOME() and then CLEOS().

5. ENT PROC CLEOL ();

   Clears from the current cursor position to the end of the line, by outputting "<esc> K".

6. ENT PROC FORCEBUFFEROUTPUT ();

   This procedure calls OUT(ETX) to force the output of OUTCL.BFR to the screen. It must be called if there is any danger of OUTCL.BFR becoming full, i.e. if more than 80 characters are output between successive newline or ETX characters.

7. ENT PROC PRINTTIME ();

   This is not a screen cursor routine. It calls TIMDAT(-1), and is only present in this module for historical reasons.

APPENDIX K

STUDENT PROJECT INSTRUCTION SHEET

It will be noticed that the project sheet on the next few pages refers to two data bases: the data-base MEDCOM and a simulation data-base. This is because when this project was set, the simulation part of MEDCOM was kept in a separate resident common called MEDSIM. The simulation data in MEDSIM was later included inside MEDCOM, enabling choice between simulation and real system to be done at attach time rather than taskbuild time.

UNIVERSITY OF CAPE TOWN

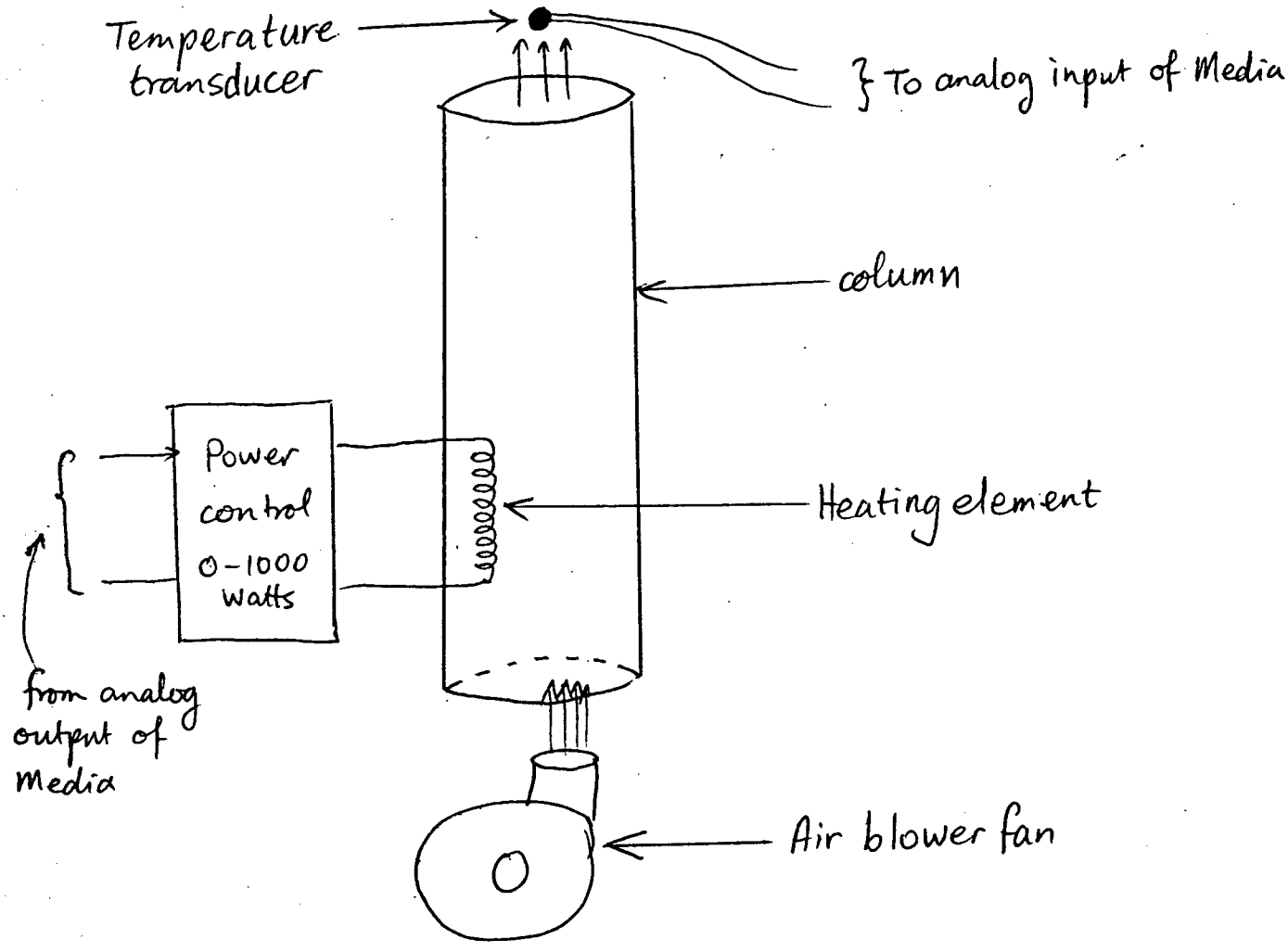
DEPARTMENT OF ELECTRICAL ENGINEERING

EE 476 REAL-TIME COMPUTER SYSTEMS PROJECT

INTRODUCTION

The project is aimed to give the student practical experience of real-time computer systems. The requirement of the project is for the student to design a software system to control the temperature of the air in a vertical column. Since the purpose of this course is to develop computer techniques rather than control techniques, a simple control algorithm (which is discussed below) is quite adequate.

The air column system is shown in the diagram below



The air is blown up the column by a blower fan, and, during its passage up the column, is heated by an electrical heating element. The temperture of the air emerging from the top of the column is measured by a temperature transducer (discussed below). The rate of flow of air is constant and cannot be varied.

The power to the heating element can be varied between 0 and 1000 Watts.

The temperature of the air emerging from the column must not be allowed to exceed 100°C.

Controlling the air-column system

The software system to control the temperature must allow the operator to change the desired temperature at any time, must provide a control algorithm for the temperature control and facilities to log the temperature on disc. Obviously all these activities must occur asynchronously and independently of each other so three tasks are required:

1. A task to control the temperature by adjusting the power to the heater.
2. A task to enable the operator to change the setpoint temperature, and to display the values relating to the column on his VDU.
3. A task to log the activity of the column onto disc.

The 3 tasks are concurrent in real-time and must all access the Media data-base.

A task, called the Media update task, performs a measurement scan of the MEDIA system at a regular interval and obtains data from the column (i.e. its temperature). The interval between measurements is known as the SAMPLING INTERVAL. Sampling is inherent in any computer control system. The results of the measurement scan are stored in a data base in the computer which is common to all tasks, i.e. all tasks can read and write to this data base (which is called MEDCOM meaning Media Common.) Tasks which need this data, such as a control task, then read MEDCOM and process the data they obtain from it. The control task would then output a value to the Media interface which would set the power to the heating element. The control task would do this at a regular interval known as the CONTROL INTERVAL.

Now four (or more) tasks are going to be accessing and manipulating the common data base (the media updating task, the control task, the OCP task and the logging task.) These tasks are running concurrently and the possibility exists that 2 tasks or more will try to use the data base simultaneously. This could lead to errors (e.g. say that the updating task updates the temperature and is then interrupted by the logging task which logs the temperature and the time, the latter having not yet been updated.) In order to avoid this a task must 'secure' the data base for uninterrupted working and then 'release' it when it is finished. If this is going to work all tasks must secure the data base before they work with it. If another task has already secured the data base the second task must wait until it has released it before continuing.

The control task and logging task must service the data base at regular intervals. After they have serviced the data base they must wait a period of real time before repeating. To do this the application program makes use of a "system directive" (i.e. a routine which interfaces to the operating system).

Simulating the system

Because there is only one set of air-column apparatus, and also because applications programs must be properly debugged before being used on a real-life system, a task has been supplied which simulates the behaviour of the column. Tasks running using the simulated system access a different data-base. This data base is updated by the simulation task instead of the media update task. The idea is that the software system will be developed and debugged using the simulation data-base, and then run using the real apparatus and data base.

As far as the software design for the project is concerned there is no difference between the actual and the simulated system. The simulation is a calculated real-time replica of the behaviour of the real column.

In order to provide a simulated real-time replica of the column behaviour the computer calculates the value of the temperature that would have been read in at each sampling interval. It does this at the same frequency as the sampling interval, so the update of the simulation common data base occurs in the same way as for the real system. Obviously the calculated result depends on the value of the setpoint temperature which is set by the OCP task. The OCP task does this by updating a memory location in the common data base. Another location stores the real time of latest update.

The Media system

The Media process control system, and the air column apparatus, is located in the University Chemical Engineering building and is connected to the PDP 11/23 computer in the Electrical Engineering building via a serial 9600 baud data line. All the link communication is performed by the Media update task described earlier.

The Media system has the following inputs and outputs:

1. 16 lines of digital output
2. 32 lines of digital input
3. 4 analogue outputs (DAC)
4. 16 analogue inputs (ADC)

The present project uses only the analogue part of the system - in the real system input and output number 1, and in the simulated system any one of input and output pairs 1, 2, 3 or 4.
The power to the heater is set by the analogue output, and the value of temperature is read via the analogue input.

Attaching the Media Outputs

If user A were to be running a program which uses analogue output number 2, say, to output the value for the power to the heating element, he must be sure that no other user can write to output number 2 - otherwise the other user could destructively interfere with the task user A is trying to perform. To prevent such an eventuality, a task is provided which 'attaches' a user to a particular output and ensures that no other user can use that output. This task, called ATTACH, must be run before executing any of the tasks which access the common data-base. When he is finished running his control tasks, the user runs ATTACH again to 'detach' from the output he had attached to. This is explained more fully later.

2. PROGRAMMING SPECIFICS

(a) Common data base - Medcom

This contains all the information pertinent to the media inputs and outputs : data values, addresses, setpoints, most recent update times, UIC's of users attached, etc. However, the user need not know exactly how the data is laid out because a number of procedures (see part (c) below) have been provided to read and write to/from Medcom.

Note: As mentioned earlier, there are in fact two copies of the database - one used for the simulated system and one for the real media system. The correct database is automatically linked in at task build time if you use $RTLCMP. (see below)

(b) Attaching to and detaching from outputs

(i) To attach type >$ATTACH
You will be prompted with a menu of choices - e.g. whether you wish to specify which output to attach to, or wish to attach to the lowest numbered output that is free, etc.

(ii) To detach, type $ATTACH and respond to the menu appropriately.

YOU WILL NOT BE ABLE TO WRITE TO AN OUTPUT WITHOUT ATTACHING YOURSELF TO ONE FIRST.
YOU MUST ALWAYS DETACH WHEN YOU ARE FINISHED.

(c) Accessing data in the Medcom data-base

Access to the data-base is provided via a set of subroutines provided for the purpose. To use these routines (which must be declared as EXT PROC's), you must have the following lines in your program:

```
LET ANALOG  =  0;
LET DIGITAL =  1;
LET INPUT   =  0;
LET OUTPUT  =  1;
```

These parameters are passed in the ADSWITCH and IOSWITCH parameters of the procedure calls.

The following procedures may be used:

(i) ENT PROC ATTACHED (INT ADSWITCH, REF ARRAY INT OUTARRAY);

You cannot know which output you will be using when you are writing an application program. In order to write to the correct output, you must use the procedure ATTACHED to find out which outputs you are attached to.
This procedure puts into the array OUTARRAY the numbers of those outputs to which the user is attached. The rest of OUTARRAY is filled with zeros. For this application we are using only analog outputs and so the ADSWITCH parameter must be given as ANALOG. The length of OUTARRAY must be greater than or equal to the number of outputs attached to (e.g. if you know that you will only attach to one output, OUTARRAY can be an array of length 1).

(ii) ENT PROC WROUTPUT (INT ADSWITCH, MDATA, CHANNUM);
This procedure writes the data MDATA to output channel number CHANNUM. In this application, ADSWITCH = ANALOG, CHANNUM must be in range 1 .. 4 and only the least significant 8 bits of MDATA are used because the DAC is 8-bit.

(iii) ENT PROC READINPUT (INT ADSWITCH, CHANNUM) INT;
This procedure reads in analogue (if ADSWITCH = ANALOG) input number CHANNUM and returns the value read. CHANNUM must be in range 1 ... 16. Since the ADC is a 10-bit ADC, the value is put in the 10 least significant bits of the returned integer, the other bits being set to zero.

(iv) ENT PROC RDSCANTIME (INT ADSWITCH, IOSWITCH, CHANNUM) REAL;
This procedure returns the time (in seconds past midnight) that the specified input or output channel was last updated by the update task.
As before we let ADSWITCH = ANALOG and CHANNUM be the channel number. IOSWITCH must be INPUT for an input or OUTPUT for an output.

(v) ENT PROC SETSETPT (INT CHANNUM, FRAC VALUE);
This procedure sets the setpoint for the analogue input controlled by analogue output number CHANNUM to VALUE.

(vi) ENT PROC GETSETPT (INT CHANNUM) FRAC;
This procedure returns the setpoint (as set up in a call to SETSETPT) of the analog input controlled by analogue output number CHANNUM.

(d) Securing and releasing the data-base

This is done automatically in the interface procedures described in (c) above, and occurs transparently to the user program.

The purpose of these routines is to open and close I/O channels. An I/O channel can be a VDU output, keyboard input, or disk file input or output. More than 2 channels can be open at any time but only two channels (one for input, one for output - see section (k) below) can be in use at one time.

Thus after you have initialised the I/O channels at the beginning of your program by saying

```
INITIO ( )
```

you open the channels you want to use, e.g.

```
OPENOUTFILE ("LOGFIL.OUT");  This opens LOGFIL.OUT as an output file.
```

Note that the input and output channels to the VDU console are automatically opened by INITIO.

You specify which channels you want to use for input and output by saying:

```
SWITTIN()  % Input currently comes from terminal %
SWITTOUT() % Output currently goes  to terminal %
SWFILOUT() % Output currently goes  to the opened output file %
```

At any time during your program you can change your current input or output channel by calling one of the above routines.

The code given in RRJOB in [2,1] EE476.RTL will open the user's console for input and output and set them as the current channels.

The command to open an output channel to the disk file (for the logging task) is also provided although it is between % signs so will not be compiled unless these are removed. The reason for this is that for tasks that do not require disk output you do not wish to open a disk file since it actually enters the file in your directory and will clutter it with a whole lot of empty files.

It is important to close a disk output file properly by executing

```
CLOSEOUTFILE()
```

when you have finished outputting to the disk.

Be careful not to output data in an endless loop to the disk because you will fill the disk!

Note: Input from files can also be done - consult the demonstrator for further details.

(k) Stream I/O

RTL/2 as a language has no equivalent of the FORTRAN READ or WRITE statements. The user program has to provide procedures to output to the device (via the I/O Driver of the operating system) or input from it. In order to be able to use the same routines for different devices one writes to the output channel and reads from the input channel rather than to the particular device. To write the same thing to different devices you can use the same output procedure and merely switch channels (using SWTTIN, SWTTOUT, and SWFILOUT).
For this reason Standard Stream I/O routines for use by RTL/2 programs are defined. A description of these standard Stream I/O routines is provided. Again these form a library which is linked to your program when your task is being built so you must provide EXT references for any routine you use.

Note: When writing to the output channel, whether text or data, your output will only actually be sent to the device when the appropriate termination character is output. Valid termination characters are LF (line feed) if a new line is required or ETX(control-C) if you wish to stay on the same line. Also, the ENQ (enquire) character should be used to terminate any text that prompts for input from the terminal console.

LF, ETX and ENQ are defined via 'let' definitions in [2,1] EE476.RTL. You cannot include these characters in a character string unless enclosed between # signs e.g.

```
TWRT("go to new line next #LF#");
```

You normally use LF except if you are prompting for input on the same line e.g.

```
TWRT ("INPUT INTEGER VALUE #ENQ#")
I: = IREAD();
```

If ENQ were not included the prompt would not appear on the screen. If you used LF the value input would be echoed on the line below your prompt.

(ℓ) Error numbers:

The meaning of the compiler error numbers (as you will see in your .SRC files) is given in the accompanying table.

If you make a logical mistake which results in program failure your task will be terminated and a message output:

'TASK' - nnn

Where TASK is the name of your task and nnn is the error number summarised in the accompanying table of run-time errors.

(m) Running the programs

In order to start your software system you run each task in turn.

For example let TASK1 = CONTROL TASK
TASK2 = OCP TASK
TASK3 = LOGGING TASK

TASK1 requires K, TI (see part (o) below) to be input and then continues controlling without further operator contact.

```
>RUN TASKI/TASK = TASK1
 INPUT K, TI : 1.0,2.0
```

After you input your values the cursor moves back to the beginning of the line but does not line feed. The system is now running TASK1. To be able to communicate with MCR you press 'return' again and you get the MCR prompt. You can then run TASK 3, etc.
To stop a task running, use the ABM command. ABM is a special version of the operating system ABO command, and must be used when aborting any tasks which access the data-base.
The format is:

```
ABM 'taskname'
```

Note: DO NOT USE 'ABO' TO ABORT MEDIA TASKS. This is because ABO does not ensure that the task being aborted releases the data-base. and thus the data-base may "hang".

(n) Printout required

Your logging task must give for every sampling interval (= 1 sec) the information about the column, i.e. time, setpoint, power, temperature. You must also show the effect of changing the setpoint. About 3 minutes of printout is sufficient.
The printout is contained in LOGFIL.OUT. Check it on your screen using PIP before actually printing it.

(o) Control interval and algorithm

The sampling interval is one second.

For the purposes of this project use a PI control algorithm given by

$$\text{POWER} = K*\left\{ e_n + \sum_n \frac{\Delta t_c}{T_I} \frac{(9e_n + 19e_{n-1} - 5e_{n-2} + e_{n-3})}{24} \right\} ,$$

where n is the error at the present sampling interval (given by temperature minus setpoint) and n-j is the error at the sampling interval j previously.

Δtc is the control interval which is taken as 1.0 seconds for the pruposes of the marked report.

Note: 1. The heater element is capable of delivering up to 1000 Watts. Thus POWER must be limited so that it is positive and does not exceed 1000 Watts.

2. It is a good idea to limit the contribution of the integral term in the above algorithm to the range - 1000 to + 1000 Watts, so that it does not grow excessively.

(p) The value to write to the analogue DAC output to produce a heating power of POWER watts (where 0.0 < POWER < 1000.0) can be determined from the following

(1) Power produced is linearly related to digital value written to the DAC (i.e. the analogue output).

(2) Digital 00 H produces 0 Watts

(3) Digital FF H produces 1000 Watts

The value ADCVAL read in via the analogue input analog-to-digital converter is however not linearly related to the actual temperature. This is because the thermistor used has a non-linear calibration curve.
The temperature can be calculated using the following relation:

temperature in degrees Centigrade = $a_3X^3 + a_2X^2 + a_1X + a_0$

where $X = \frac{1}{4}$(ADCVAL)
$a_3 = 1.1635\ E-5$
$a_2 = -2.8494\ E-3$
$a_1 = 0.5916$
$a_0 = 17.367$

(q) The task MEDRMD

Typing RUN MEDRMD causes a display of the activity of the data-base to the screen. The information displayed includes which user is attached to which output, when last the data-base was updated and which task has currently secured the data-base.

APPENDIX L

ERROR NUMBERS

Error numbers for all the software modules are listed below.

A prefix of 'U' indicates an unrecoverable error (a call to RRGEL) and a prefix of 'R' indicates a recoverable error (a call to ERP).

L.1 Errors reported by MEDUSER

R715 : User attempted to write to a part of MEDCOM (an output) to which he was not attached.
R716 : ADSWITCH or IOSWITCH parameter was not Ø or 1.
R717 : Channel number requested does not exist.
R7Ø9 : Media input/output is out of scan.

L.2 Errors reported by MEDLNK

R5Ø1 : BCC error in frame received from Media.
R5Ø2 : Alignment error in frame received from Media
R5Ø3 : Media code in reply frame from Media does not correspond to the command code sent to Media.

L.4 Errors reported by PLOTLIB

U609 : XMIN > XMAX or YMIN > YMAX in call to SCALE.
R610 : Error in reading the position of the crosshairs.
R611 : Attempt to MOVE or DRAW to a point off the edge of the screen.

L.5 Errors reported by AST

U607 : ASTX$S directive failure.
U608 : Other directive failures.

L.6 Errors reported by GSECREL

R215 : During facility release, state of event flag not consistent with facility having been secured.
U14 : Illegal event flag number (not in range 33..56).
U15 : Attempt to release a facility which was not secured.
U16 : Attempt to secure a facility that the task has already secured.

L.7 Errors reported by ODT

R604 : Unexpected end-of-file while reading SMT.TSK from disk.
R605 : Failure to open, close or read from the files SMT.TSK or SMTLOAD.ABS.

APPENDIX M

MEDIA SYSTEM STARTUP PROCEDURE

This appendix describes the procedure to be followed when the Media system is started up.

M.1 Starting up the GEC Media system

1. Plug the serial lead from the terminal into its socket on the serial line interface unit and sign on to the PDP.

2. Plug the Micro-Media serial line into its socket on the interface unit. This line will have been set up at system startup to slave, no-echo, read-pass-all and 9600 baud.

3. Power up the Media bins.

4. Run the Media update task MEDUPDAT.

5. Run MEDRMD to check that all is well.

M.2 Starting up the LSI-11 Media system

1. Plug the serial lead from the terminal into its socket on the serial interface unit and sign on to the PDP.

2. Plug the LSI-11 console line lead into its socket on the interface unit. This line will have been set up at system startup to slave, no-echo, read-pass-all and 9600 baud.

3. Power up the LSI and the Media bins.

4. Restart the LSI by raising the 'reset' switch while the 'halt' switch is in the up position. Then drop the 'halt' switch and raise it again.

5. Run the ODT program from the terminal.

6. type 'F'<CR> to flush the buffer. and then type a few ODT commands to check that communications between the terminal, PDP and LSI are functional.

7. Use the 'L' command to load and run the SMT Micro-Media replacement program. (See the description of the ODT program in chapter 3).

8. Exit ODT by typing control-Z.

9. Run the Media update task MEDUPDAT. The entire system should now be functional and ready for user applications programs to run.

10. Run MEDRMD to check that all is well.

APPENDIX N

SOFTWARE LISTINGS: HOST SYSTEM

Listings of all the software modules which run on the PDP-11/23 host system are given here. Pages of the listing have a circled page number at the top right-hand side of the page. As an example, the 3rd page of the 2nd program in this appendix is numbered "N-2-3".

The following are the programs listed in this appendix (given in roughly the same order as their discussion in the main text):

```
OPTION (1) CM;
TITLE ODT;
```

N-1-1

```
%                                                                    %
% THIS ROUTINE IS USED TO COMMUNICATE WITH THE LSI 1123 ODT          %
% AND DOWNLOAD THE SMT.TSK MEDIA EMULATION SOFTWARE INTO THE LSI.    %
%                                                                    %

LET LF = OCT 012;                  % LINE FEED CHARACTER             %
LET CR = OCT 015;                  % CARIAGE RETURN CHARACTER        %
LET ETX = OCT 003;
LET ENQ = OCT 005;
LET EOS = OCT 200;                 % END OF STREAM CHARACTER         %
LET BEL= OCT 007;                  % BELL CHARACTER                  %
LET EOM = OCT 200;                 % END OF FILE CHARACTER           %
LET SP = OCT 040;                  % SPACE CHARACTER                 %
LET DOL = HEX 24;                  % DOLLAR CHARACTER                %
LET     RNE      =   OCT 1020;     % CODE FOR READ NO ECHO           %
LET     RAL      =   OCT 1010;     % CODE FOR READ ALL               %
LET     TMO      =   OCT 200;      % CODE FOR TIMEOUT                %
LET     WRTCODE  =   OCT 410;      % WRITE WITH WAL                  %
LET     KILCODE  =   OCT 0012;     % KILL ALL I/O                    %
LET     DETCODE  =   OCT 2000;     % DETACH DEVICE                   %
LET     ATTCODE  =   OCT 1400;     % ATTACH                          %
LET     SERIALWRITELUN  =   5;     % LU OF SERIAL WRITE              %
LET     SERIALREADLUN =   6;       % LU OF SERIAL LINE               %
LET     INFILELUN       =   3;     % UNIT NUMBER OF INPUT FILES      %
LET     YES      =   1;
LET     NO       =   0;

 % I HAVE COINED THE FOLLOWING NAMES FOR SOME OF THE STATES OF       %
 % THE ODT.   SEE PROCESSOR HANDBOOK P 80. NUMBERS CHOSEN FOR       %
 % THE STATES/MODES ARE ARBITRARY.                                   %

LET     START    =   1;            % START STATE OF 11/23'S ODT      %
LET     MEMORY   =   2;            % MEMORY STATE                    %
LET     REGSTR   =   3;            % REGISTER STATE                  %
LET     PSW      =   4;            % PSW STATE                       %

%   MODE DEFINITIONS.                                                %
%   =================                                                %

MODE IOSTAT(BYTE IOSTLOW,                % IOSTATUS LOW BYTE         %
                 IOSTHIGH,               % IOSTAT HIGH BYTE          %
            INT IOSTVAL);                % IOSTAT VALUE              %

%   EXTERNAL PROCEDURE DEFINITIONS.                                  %
%   ===============================                                  %

%   THE FOLLOWING ARE STANDARD STREAM I/O PROCEDURES - REFER         %
%   TO THE STREAM I/O MANUAL                                         %
EXT PROC ()TTIO;
EXT PROC(REF ARRAY BYTE) TWRT;
EXT PROC (REF ARRAY BYTE,REF ARRAY BYTE)INT TREAD;
EXT PROC () INT IREAD, OREAD;
EXT PROC (INT,INT,INT) MARKTIME;
EXT PROC (INT) IWRT ,OWRT;
EXT PROC (INT) CANMARK, RESET;
EXT PROC (INT)RRGEL;
EXT PROC (INT,REF ARRAY BYTE)ASSIGN;
EXT PROC () RRNUL;
EXT PROC (INT,                           % I/O FUNCTION CODE      %
```

```
        INT,                                % LUN OF DEVICE          %
        INT,                                % EVENT FLAG NUMBER      %
        INT,                                % PRIORITY               %
        REF IOSTAT,                         % I/O STATUS BLOCK       %
        PROC(),                             % AST SERVICE ROUTINE    %
        REF ARRAY INT) RSXQIW, RSXQIO;      % DEVICE DEPENDENT       %
                                            % PARAMETERS             %

%   LOCAL DATA BRICK DEFINITIONS.                                     %
%   =============================                                     %

DATA LOCAL;
    REF ARRAY BYTE TERMB:="#CR,LF,EOS#";% TERMINATING CHARACTERS%
                                        % FOR INPUT              %
    INT STATE := START;                  % STATE OF 11/23 ODT    %
    INT TIMEOUT := YES;                 % TIMEOUT ON SERIAL READ%
    INT  ACCEPT :=YES ;                 % FLAG FOR COMMAND OK    %
    INT  NIN := 0;                      % NUMBER OF CHARACTERS   %
                                        % EXPECTED FOR READ      %

    INT  RECEND := FILEND := 0;         % FLAGS FOR END OF       %
                                        % RECORD AND FILE,       %
    INT  NCHARS:=0;                     % NUMBER OF CHARACTERS   %
    INT  NREAD:=0;                      % NO OF CHARS READ BY    %
                                        % SERIALR().             %
    BYTE BCC;                           % BLOCK CHECK CHAR       %
    IOSTAT STATUS;                      % IO STATUS BLOCK        %
    ARRAY (7) BYTE IBLK;                % USED BY LOADLOADER() TO HOLD  %
    REF ARRAY BYTE  BUFP := IOBUF;      % POINTER TO IOBUF       %

    ARRAY (6) INT  DEVPARM := (0,0,0,0,0,0);    % DEVICE DEPENDENT      %
    REF ARRAY INT  PP := DEVPARM;               % PARAMETERS            %
ENDDATA;

%   MAIN PROCEDURE = RRJOB.                                           %
%   =======================                                           %

ENT PROC RRJOB();
    ASSIGN(1,"TI:");
    ASSIGN(2,"TI:");
    INITIALISEIO();                             % INITIALIZE STREAM I/O %
    SWTTIN();
    SWTTOUT();

    ASSIGN(INFILELUN,"SY:");            % LUN 3 IS THE LUN FOR FILE IN  %
    ASSIGN(SERIALREADLUN,"TT10:");
    ASSIGN(SERIALWRITELUN,"TT10:");

    TWRT("#CR,LF#LSI 11 CONSOLE EMULATOR  VER 3.1 5/4/83#CR,LF#");

    STATE := START;         % INITIALISE STATE  %

       % INITIALIZE THE BUFFERS AND CONTROL FOR SERIAL IO      %

    FOR I:=1 TO 6 DO DEVPARM(I):=0; REP;
    RSXQIW(ATTCODE,SERIALREADLUN,1,0,STATUS,RRNUL,PP);
    TWRT("STATUS AFTER ATTACH #CR,LF#");
    TWRT("LOWBYTE /"); IWRT(STATUS.IOSTLOW); TWRT("#LF#");

       % LOOP TO READ FROM TERMINAL, WRITE TO LSI 1103, READ   %
       % THE ANSWER AND DISPLAY IT ON THE TERMINAL              %
```

```
          FOR I:=1 TO 6 DO DEVPARM(I):=0;REP;
          RSXQIW(KILCODE,SERIALREADLUN,1,0,STATUS,RRNUL,PP);
        ELSE
          NREAD := STATUS.IOSTVAL;
          CANMARK(1);   % CANCEL MARKTIME REQUEST        %
        END;

        IF STAT < 0 THEN TWRT("READ ERROR: STATUS ");
                IWRT(STAT); TWRT("#CR,LF#");
        END;
    ELSE
        NREAD:=0;                % NO CHARS READ          %
    END;

ENDPROC;

PROC GETNREAD()INT;
  % FINDS OUT, IN CASES OF TIMEOUT ON INPUT, HOW MANY CHARS WERE          %
  % RECEIVED BEFORE TIMEOUT. DOES THIS IN A CRUDE WAY: WORKS OUT HOW     %
  % MANY CHARS IN IOBUF BEFORE FIRST OCCURRENCE OF A NULL, WHICH IT IS   %
  % ASSUMED THE LSI WILL NEVER SEND WHILE IN CONSOLE ODT MODE.           %
  FOR I:=1 TO 80 DO
    IF IOBUF(I)=0 THEN
      RETURN(I);
    END;
  REP;
  RETURN(80);   % MORE THAN 80 CHARS IN BUFFER; SHOULD NEVER OCCUR       %
ENDPROC;

PROC SERIALW ();
    INT J,STAT;
    IF NCHARS > 0 THEN
        FOR J:=1 TO 6 DO DEVPARM(J):=0; REP;
        DEVPARM(1) := BUFSET(IOBUF);
        DEVPARM(2) := NCHARS;
        RSXQIW(WRTCODE,SERIALWRITELUN,1,0,STATUS,RRNUL,PP);
        STAT:=CHECKST();
        IF STAT<0 THEN
          TWRT("WRITE ERROR: DIRECTIVE STATUS ");
          IWRT(STAT);TWRT("#LF#");
        END;
    END;
ENDPROC;

PROC CHECKST()  INT;

        % THIS PROCEDURE CHECKS THE IO STATUS.  %
        % IT RETURNS THE STATUS VALUE           %
        % -VE RETURNED VALUE MEANS A FAILURE    %

    INT  STTUS;

    STTUS :=  (INT(STATUS.IOSTLOW) SLL 8) SRA 8;
    RETURN (STTUS);
ENDPROC;

PROC SCAN();

        % THIS ROUTINE SCANS THE INPUT COMMAND FOR A VALID      %
        % COMMAND CODE SYNTAX.  THE ACCEPT FLAG IS SET TO YES   %
        % IF THE COMMAND IS VALID ELSE IT IS SET TO 1.          %
```

```
    IF IOBUF(1) = 'F' THEN
        FLUSH();
    ELSEIF IOBUF(1) = 'H' THEN
        HELP();
    ELSEIF STATE = START THEN
        IF IOBUF(1) = DOL OR IOBUF(1) = 'R' THEN
            DOLLAR();
        ELSEIF IOBUF(NCHARS) = '/' THEN
            SLASH();
        ELSEIF IOBUF(NCHARS) = 'G' THEN
            GO();
        ELSEIF IOBUF(NCHARS) = 'P' THEN
            PROCED();
        ELSEIF IOBUF(1) = 'L' THEN
            LOAD();
        ELSE
            ACCEPT:= NO;
        END;
    ELSE                              % STATE IS REGSTR, MEMORY OR PSW %
        IF IOBUF(NCHARS) = 'C' THEN
            CRETURN();
        ELSEIF IOBUF(NCHARS) = 'N' THEN
            LFPROC();
        ELSE
            ACCEPT := NO;
        END;
    END;
    NIN := NIN + NCHARS;
ENDPROC;

PROC DOLLAR();

        % THIS ROUTINE IS USED FOR THE INTERNAL REGISTER OPEN    %
        % COMMAND                                                %

    IF NCHARS # 3 THEN
        ACCEPT:= NO;
    ELSE
        IF IOBUF(3) # '/' THEN
            ACCEPT := NO;
        ELSE
            IF IOBUF(2) < '0' OR IOBUF(2) > '7' THEN
                IF IOBUF(2) = 'S' THEN  % WANTING TO OPEN PSW   %
                    ACCEPT :=YES;
                    NIN := 7;
                    TIMEOUT := YES;
                    STATE := PSW;
                ELSE
                    ACCEPT := NO;
                END;
            ELSE                              % WANTING TO OPEN REGISTER %
                ACCEPT := YES;
                NIN    := 7;
                TIMEOUT := YES;
                STATE   := REGSTR;
            END;
        END;
    END;
ENDPROC;
```

```
PROC SLASH();

        % THIS ROUTINE CHECKS THE OPEN MEMORY COMMAND    %

    IF NCHARS >=2 AND NCHARS <= 7 THEN
        ACCEPT := YES;
        FOR I:= 1 TO NCHARS - 1 DO
            IF IOBUF(I) < '0' OR IOBUF(I) > '7' THEN
                ACCEPT := NO;
            END;
        REP;
    ELSE
        ACCEPT:= NO;
    END;
    IF ACCEPT = YES THEN
        NIN := 7;
        TIMEOUT := YES;
        STATE := MEMORY;
    END;
ENDPROC;

PROC FLUSH();

        % THIS ROUTINE IS USED TO FLUSH THE INPUT BUFFER %
    IF NCHARS = 1 THEN
        NCHARS:=0;
        NIN := 80;
        ACCEPT := YES;
        TIMEOUT := YES;
        STATE  := START;
    ELSE
        ACCEPT := NO;
    END;
ENDPROC;

PROC HELP();

    % THIS ROUTINE OUTPUTS TO TI: A DESCRIPTION OF HOW TO USE   %
    % THE PROGRAM.                                              %
    % DOES NOT CHANGE 'STATE'.                                  %

    IF NCHARS=1 OR IOBUF(2)='E' AND IOBUF(3)='L' AND IOBUF(4)='P' THEN
        TWRT("#LF(6)#THIS PROGRAM TAKES ODT COMMANDS AND SENDS THEM DOWN THE
        TWRT("#LF#SERIAL LINE TO THE LSI-11 CONNECTED TO THE SERIAL LINE.");
        TWRT("#LF#IN ADDITION, IT CAN LOAD A BINARY LOADER PROGRAM");
        TWRT("#LF#SMTLOAD.ABS WHICH CAN THEN DOWNLOAD THE FILE SMT.TSK.");
        TWRT("#LF#THE COMMANDS ARE THE SAME AS ODT COMMANDS EXCEPT THAT YOU"
        TWRT("#LF#MUST USE 'N' (NEXT) INSTEAD OF LINE FEED AND 'C' INSTEAD O-
        TWRT("#LF#CARRIAGE RETURN.#LF#");
        TWRT("#LF#NON-ODT COMMANDS ARE :#LF#");
        TWRT("#LF#'F' : FLUSH THE INPUT BUFFER (10 SECOND TIMEOUT).");
        TWRT("#LF#'L' : LOAD : PROMPTS THE USER WITH QUESTIONS RE DOWNLOADIN-
        TWRT("#LF#CONTROL-Z : EXIT TO MCR.");
        TWRT("#LF#'H' OR 'HELP' : PRINTS THIS TEXT.#LF#");

        NCHARS:=0;
        NIN:=0;
        ACCEPT:=YES;
        TIMEOUT:=YES;   % THIS LINE REDUNDANT,BUT SO WHAT.      %
    ELSE
        ACCEPT:=NO;
    END;
```

```
ENDPROC;

PROC CRETURN();

        % THIS ROUTINE DEALS WITH THE <CR> COMMAND                %

    IOBUF(NCHARS) := CR;
    IF NCHARS = 1 THEN
        ACCEPT := YES;
    ELSEIF NCHARS <= 7 THEN
        ACCEPT := YES;
        FOR I:= 1 TO NCHARS - 1 DO
            IF IOBUF(I) < '0' OR IOBUF(I) > '7' THEN
                ACCEPT := NO;
            END;
        REP;
    ELSE
        ACCEPT := NO;
    END;
    IF ACCEPT = YES THEN
        IOBUF(NCHARS) := CR;
        NIN := 3;
        TIMEOUT:= YES;
        STATE := START;
    END;
ENDPROC;


PROC LFPROC();

        % CLOSE LOCATION AND OPEN NEXT LOWER ONE                  %

    IF NCHARS = 1 THEN
        ACCEPT :=YES;
    ELSEIF NCHARS <= 7 THEN
        ACCEPT := YES;
        FOR I:= 1 TO NCHARS - 1 DO
            IF IOBUF(I) < '0' OR IOBUF(I) > '7' THEN
                ACCEPT := NO;
            END;
        REP;
    ELSE
        ACCEPT := NO;
    END;
    IF ACCEPT = YES THEN
        TIMEOUT := YES;
        IOBUF(NCHARS) := LF;
        IF STATE = MEMORY THEN
            NIN := 15;
        ELSEIF STATE = REGSTR THEN
            NIN := 11;
        ELSE                        % STATE IS PSW %
            NIN := 2;
            STATE := START;
        END;
    END;
ENDPROC;

PROC GO();

        % GO FROM GIVEN POINT   %
```

```
    IF NCHARS = 1 THEN
        ACCEPT := YES;
    ELSEIF NCHARS <= 7 THEN
        ACCEPT := YES;
        FOR I:= 1 TO NCHARS - 1 DO
            IF IOBUF(I) < '0' OR IOBUF(I) > '7' THEN
                ACCEPT := NO;
            END;
        REP;
    ELSE
        ACCEPT := NO;
    END;
    IF ACCEPT = YES THEN
        NIN := 11;
        TIMEOUT := NO;          % NO TIMEOUT ON INPUT    %
        STATE := START;
    END;
ENDPROC;


PROC PROCED();

        % PROCEED %

    IF NCHARS # 1 THEN
        ACCEPT := NO;
    ELSE
        ACCEPT:= YES;
        NIN := 11;
        TIMEOUT := NO;          % NO TIMEOUT ON INPUT    %
        STATE := START;
    END;
ENDPROC;


PROC LOAD();

        % THIS PROC PERFORMS THE LOAD INSTRUCTION        %
    IF NCHARS = 1 THEN
        TWRT("#LF#DOWN LOAD THE ABSOLUTE LOADER?  Y/N    #ENQ#");
        NIN := TREAD(IOBUF,TERMB); TWRT("#LF#");
        IF IOBUF(1) = 'Y' OR IOBUF(1)='y' THEN
            LOADLOADER();
        END;

        TWRT("#LF,LF#ENSURE HALT SWITCH ON LSI IS UP BEFORE LOADING.");
        TWRT("#CR,LF#LOAD SMT ?  Y/N    #ENQ#");
        NIN := TREAD(IOBUF,TERMB); TWRT("#LF#");
        IF IOBUF(1) = 'Y' OR IOBUF(1)='y' THEN
            LOADSMT();
        END;

        TWRT("#LF#START SMT RUNNING ? Y/N   #ENQ#");
        TREAD(IOBUF,TERMB); TWRT("#LF#");
        IF IOBUF(1) = 'Y' OR IOBUF(1)='y' THEN
            NCHARS := 2;        % 2 CHARS IN 'OG'        %
            NIN :=2;            % EXPECT 'OG' IN REPLY   %
            IOBUF(1):='O';IOBUF(2):='G';
            SERIALW();
            SERIALR();
            FOR I:=1 TO NREAD DO
```

```
            OUT(IOBUF(I));  % SEND REPLY TO TI:     %
        REP;
    END;

    NCHARS := 0;
    NIN := 0;
    TIMEOUT := YES;          % THIS LINE IS NOT NEEDED       %
    ACCEPT := YES;
ELSE
    ACCEPT := NO;
END;
ENDPROC;

PROC LOADLOADER();

    % THIS PROC LOADS THE ABSOLUTE LOADER.  THE LOADER IS IN        %
    % THE ASCII FILE WITH UNIT NUMBER INLD.                         %


INT NRD:=0;
INT SKIP:=7;
INT K:=0;
BYTE INB := 0;

OPENINFILE("SMTLOAD.ABS");

PROCESS("157000/",7);        % OPEN THE FIRST LOCATION       %
SWFILIN();                   % OPEN THE ABSOLUTE LOADER      %
                             % INPUT STREAM                  %
WHILE INB # EOS DO
    INB := IN();             % GET NEXT BYTE                 %
    IF INB = LF THEN SKIP := 7; END;
    IF SKIP > 0 THEN
        SKIP := SKIP -1;
    ELSE
        IF INB # SP THEN
            K := K + 1;
            IBLK(K) := INB;
            IF K = 6 THEN
                K := 0;
                IBLK(7) := 'N';
                PROCESS(IBLK,7);
            END;
        END;
    END;
REP;
SWTTIN();
PROCESS("C",1);          %  GET BACK INTO START STATE %
CLOSEINPUTFILE();
ENDPROC;

PROC PROCESS(REF ARRAY BYTE INP, INT NCH);

    % THIS ROUTINE PROCESSES ONE ODT COMMAND.  THE COMMAND  %
    % IS GIVEN IN THE IOBUF ARRAY AND IT HAS NCHARS CHARACTERS%


NCHARS := NCH;
FOR I:= 1 TO NCHARS DO IOBUF(I) := INP(I); REP;

SCAN();
IF ACCEPT = YES THEN
```

```
        SERIALW();
        SERIALR();
        FOR I:=1 TO NREAD DO
            OUT(IOBUF(I));      % SEND REPLY TO TI:      %
        REP;
    END;
ENDPROC;

PROC LOADSMT();

        % THIS PROC LOADS THE SMT OPERATING SYSTEM DOWN THE      %
        % SERIAL LINE IN ABSOLUTE LOADER FORMAT.  THAT IS        %
        % THE FIRST TWO BYTES ARE THE NUMBER OF BYTES TO FOLLOW  %
        % THE REST ARE DATA BYTES THAT ARE LOADED INTO THE       %
        % LSI 1103 IN SUCCESSIVE BYTES STARTING AT ZERO          %

    INT FILELEN,NBYTES, NREMAINDER, LSB:=0, MSB:=0, NBLKS, J, J1, J2;
    INT BLKCNT;
    INT ABORT;

    OPENBLKINFILE("SMT.TSK");

    READBLK();

    %MAX ADDRESS IS IN BYTES 11 & 12 OF FIRST RECORD OF *.TSK    %
    % FILES. I FOUND THIS OUT BY DUMPING A COUPLE OF *.TSK FILES%

    LSB:=IOBUF(11);             % LS BYTE OF MAX ADDRESS         %
    MSB:=IOBUF(12);             % MS BYTE                        %
    FILELEN := LSB + ( MSB SLL 8 ) + 1;  % LENGTH OF FILE = MAX ADDR + 1 %
    NBLKS := FILELEN SRL 9;     % 512 (I.E. 2**9) BYTES PER BLOCK %
    NREMAINDER:= FILELEN MOD 512;
    IF NREMAINDER#0 THEN
        NBLKS:=NBLKS + 1;       % SEND DATA IN COMPLETE BLOCKS ONLY      %
    END;
    NBYTES:= NBLKS SLL 9;       % NO OF CODE BYTES TO XFER (2**9 BYTES/BLOCK

    TWRT("#LF#FILE LENGTH IN BYTES (OCTAL) : "); OWRT(FILELEN);
    TWRT("#LF#");
    TWRT("NO OF 512-BYTE BLOCKS TO TRANSFER (OCTAL) : "); OWRT(NBLKS);
    TWRT("#LF#TOTAL NO OF BYTES TO BE SENT : "); OWRT(NBYTES+2); TWRT("#LF#"

%   PUT 340 INTO PSW TO DISABLE ALL INTERRUPTS  %
%   PUT START ADDRESS (157000) INTO PC.         %
    PROCESS("R7/",3);
    PROCESS("157000C",7);
    PROCESS("RS/",3);
    PROCESS("340C",4);

    NCHARS := 1;            % 1 CHAR IN 'P'          %
    % OUTPUT "P" TO RUN THE LOADER      %
    IOBUF(1):='P';
    SERIALW();  % COMMAND TO LSI TO RUN THE LOADER      %
    NIN:=1;
    TIMEOUT:=YES;
    SERIALR();
    FOR I:=1 TO NREAD DO
        OUT(IOBUF(I));  % OUTPUT THE REPLY TO TI:       %
    REP;

    % THE FIRST 2 BYTES SENT TO THE LSI GIVE THE LENGTH OF THE FILE TO  %
    % FOLLOW.                                                           %
```

N-1-10

```
    BCC:=0;
    FOR I := 1 TO 512 DO
        BCC:= BYTE(INT(BCC) + INT(IOBUF(I)));
    REP;

ENDPROC;

PROC BUFSET(REF ARRAY BYTE BUFFER) INT;

% THIS PROC TAKES AS INPUT A REF ARRAY BYTE AND GIVES AS OUTPUT AN INT  %
% EQUAL TO THE ADDRESS OF THE FIRST BYTE OF THE ARRAY POINTED TO BY     %
% BUFFER.                                                               %

    INT TEMP;
    CODE 10,0;
        MOV     *BUFFER(%5),*TEMP(%5)
        INC     *TEMP(%5)                ;SINCE 1ST BYTE IS LENGTH OF ARRAY
    *RTL;
    RETURN(TEMP);
ENDPROC;

%   SVC DATA BRICK DEFINITIONS.                                         %
%   ===========================                                         %

%   SVC DATA BRICKS ARE PART OF THE INTERFACE TO THE OPERATING  %
%   SYSTEM                                                      %

SVC DATA RRSIO;
      PROC()BYTE IN;            % STREAM I/O INPUT PROCEDURE    %
      PROC(BYTE) OUT;           % STREAM I/O OUTPUT PROCEDURE   %
ENDDATA;

SVC DATA RRSED;
      BYTE TERMCH,
           IOFLAG;
ENDDATA;

SVC DATA RRERR;
      LABEL ERL;
      INT ERN;
      PROC(INT) ERP;            % THIS IS THE ERROR EXIT PROCEDURE RRGEL %
ENDDATA;

SVC DATA RRERRX;                % THIS IS FOR DIAGNOSTIC INFO   %
      INT LINENO;
      BYTE UEFLAG,ERRLUN;
      INT RSXDSW;               % DSW RESULT FROM EXECUTIVE DIRECTIVES  %
ENDDATA;


% FILE I/O SECTION %

MODE FNBLK(INT DEVDEVN,REF BYTE DEVT,INT DIRN,REF BYTE DIRT,
       INT NAMN,REF BYTE NAMT);
MODE IOCL(REF ARRAY BYTE BFR,INT N,DV,PTR,MD,TRM);

EXT PROC(REF FNBLK,INT,INT,INT)INT FOPENB;
EXT PROC(REF INT,INT,INT,INT)INT GETBLK;
EXT PROC(REF ARRAY INT) FINIT;
EXT PROC(REF ARRAY BYTE,REF FNBLK)FNAME;
EXT PROC(INT) INT FCLOSE;
```

N-1-13

```
EXT PROC (REF FNBLK,INT,INT)INT FSTRTIN;
EXT PROC(BYTE)TTOUT,GPOUT;
EXT PROC()BYTE TTIN,GPIN,INF;

EXT DATA FILE01; REF ARRAY INT FDBTBL; ENDDATA;

SVC DATA RRCHAN; REF IOCL INCL,OUTCL;ENDDATA;

DATA FILEIOLOCAL;
  FNBLK FILENAM:=(0,DUMMY,0,DUMMY,0,DUMMY);
  BYTE DUMMY;
  IOCL TTINCL :=(TTINBUF,0,1,0,0,0);
  IOCL TTOUTCL:=(TTOUTBUF,0,2,1,0,0);
  IOCL INFILECL:=(INFBUF,0,0,0,0,0);
  ARRAY(132)BYTE INFBUF:=(' '(132));
  ARRAY(132)BYTE TTINBUF:=(0(132));
  ARRAY(132)BYTE TTOUTBUF:=(LF,' '(131));
  ARRAY(512)BYTE IOBUF;
  INT DUMINT;
  REF INT BUFADR:=DUMINT;       % WILL LATER POINT TO IOBUF     %
ENDDATA;

PROC INITIALISEIO();
  REF ARRAY BYTE IOBUFPTR:=IOBUF;
  FINIT(FDBTBL);
  % GET ADDRESS OF START OF IOBUF INTO THE REF INT BUFADR %
  CODE 10,0;
        MOV     *IOBUFPTR(5),*BUFADR/FILEIOLOCAL
        INC     *BUFADR/FILEIOLOCAL                     ;POINT TO START
  *RTL;
ENDPROC;

PROC SWTTIN();
  INCL:=TTINCL;
  IN:=GPIN;
ENDPROC;

PROC SWTTOUT();
  OUTCL:=TTOUTCL;
  OUT:=GPOUT;
ENDPROC;

PROC OPENINFILE(REF ARRAY BYTE FILETEXT);
  FNAME(FILETEXT,FILENAM);
  SWFILIN();
  IF FSTRTIN(FILENAM,1,INFILELUN) #0 THEN
    RRGEL(605);
  END;
ENDPROC;

PROC CLOSEINPUTFILE();
  IF FCLOSE(1)#0 THEN RRGEL(605);END;
ENDPROC;

PROC SWFILIN();
  INCL:=INFILECL;
  IN:=INF;
ENDPROC;

PROC OPENBLKINFILE(REF ARRAY BYTE FILETEXT);
  FNAME(FILETEXT,FILENAM);
  IF FOPENB(FILENAM,1,INFILELUN,0)#0 THEN RRGEL(605);END;
```

```
TITLE MEDFRAME;

% DIAGNOSTIC TO BUILD UP ,SEND AND RECEIVE MEDIA DATA-LINK      %
% PROTOCOL FRAMES.  IT CALLS THE LINKLB ROUTINE 'MESSANS' TO    %
% WRITE THE FRAME OUT AND RECEIVE THE REPLY FROM MEDIA.         %

LET     LF      =       OCT 012;
LET     ENQ     =       OCT 005;
LET     ETX     =       OCT 003;

EXT PROC(REF ARRAY BYTE)TWRT;
EXT PROC()INT IREAD,OREAD;
EXT PROC(INT) IWRT,OWRT;
EXT PROC(REF ARRAY BYTE,INT,REF ARRAY BYTE,INT)MESSANS;
EXT PROC() TTIO;

DATA LOCAL;
    ARRAY(80) BYTE OUTBUF;
    ARRAY(80) BYTE INBUF;
    INT NOUT;
    INT NIN;
    INT ADDR;
    INT SAVEADDR;
    INT NITEMS;
    INT DAT;
    INT GOFLAG :=0;
    INT N;
    INT COMMAND;
ENDDATA;

ENT PROC RRJOB();
    TTIO();

START:
    TWRT("#LF# MEDFRAME (LINKLB tester) (Media protocol tester) #LF,LF#");
    TWRT("1.   SINGLE MEDIA READ#LF#");
    TWRT("2.   SINGLE LIST READ#LF#");
    TWRT("3.   BLOCK MEDIA READ#LF#");
    TWRT("4.   BLOCK LIST READ#LF#");
    TWRT("5.   DIGITAL CHANGE WORDS#LF#");
    TWRT("6.   MEDIA STATUS WORD#LF#");
    TWRT("7.   WRITE TO MEDIA#LF#");
    TWRT("8.   WRITE TO LIST#LF#");
    TWRT("9.   GO#LF,LF#");
    TWRT("10.  SYNCHRONISE#LF#");
    TWRT("=>#ENQ#");

    N := IREAD();

    IF N=1 OR N=2 THEN  % SINGLE READ   %
        COMMAND := N;
        NOUT := 4;
        NIN := 6;
        ADDRESS();
        OUTBUF(1):= (BIN 01000000) LOR (BYTE(COMMAND))
    ELSEIF N=3 OR N=4 THEN      % BLOCK READ    %
        COMMAND := N;
        NOUT := 5;
        TWRT("#LF#NO OF ITEMS#ENQ#");
        NITEMS := IREAD();
        OUTBUF(4) := BYTE(NITEMS);
        NIN := 3*(NITEMS + 1);
```

```
        ADDRESS();
        OUTBUF(1) := (BIN 01000000) LOR (BYTE(COMMAND));
    ELSEIF N=5 THEN     %DIG CH WORDS   %
        NOUT :=4;
        OUTBUF(1) := BIN 01000111;
        OUTBUF(2) := 0;
        OUTBUF(3) := 0;
        NIN :=9;
    ELSEIF N=6 THEN     % READ STATUS WORD      %
        OUTBUF(1) := BIN 01001000;
        OUTBUF(2) := 0;
        OUTBUF(3) := 0;
        NOUT :=4;
        NIN := 6;
    ELSEIF N=7 OR N=8 THEN      % WRITE %
        COMMAND := N + 5;
        OUTBUF(1) := (BIN 01000000) LOR (BYTE(COMMAND));
        ADDRESS();
        NOUT :=7;
        NIN := 7;
        TWRT("#LF#DATA(OCTAL)#ENQ#");
        DAT := OREAD();
        ENCODE(DAT);
    ELSEIF N=9 THEN     % GO MESSAGE    %
        NOUT := 4;
        NIN := 3;
        OUTBUF(1) := BIN 01001110;
        GOFLAG :=1;
        ADDRESS();
    ELSEIF N=10 THEN    % SYNCHRONISE   %
        NOUT := 1;
        NIN := 20;
        OUTBUF(1) := BIN 01001010       % A SPARE CODE  %
    ELSE
        GOTO START;
    END;

    PUTBCC();

    TWRT("#LF#OUTPUT INFO :#LF#");
    FOR I:=1 TO NOUT DO
        OWRT(INT(OUTBUF(I))); TWRT(" ");
    REP;

    MESSANS(OUTBUF,NOUT,INBUF,NIN);

    TWRT("#LF#ANSWER RECEIVED :");
    FOR I := 1 TO NIN DO
        IF ((I-1) MOD 8)=0 THEN TWRT("#LF#");END;
        OWRT(INT(INBUF(I))); TWRT(" ");
    REP;
    TWRT("#LF#");

    GOTO START;

ENDPROC;

PROC PUTBCC();
    INT T:=0;
    BYTE B;
    FOR I:=1 TO NOUT-1 DO
        T:=T NEV OUTBUF(I);
```

N-14-2

```
    REP;
    B:= BYTE(T LAND OCT 077);    % CONVERT TO BYTE          %
    B:= B LOR BIN 01000000;
    OUTBUF(NOUT) := B;

ENDPROC;

PROC ADDRESS();
    IF GOFLAG = 1 THEN
        ADDR := SAVEADDR;
    ELSE
        TWRT("#LF#Media/list address #ENQ#");
        ADDR := OREAD();
    END;
    OUTBUF(2) := BYTE(ADDR LAND BIN 00001111);
    OUTBUF(3) := BYTE((ADDR SRL 4) LAND BIN 00111111);
    SAVEADDR := ADDR;
    GOFLAG := 0;
ENDPROC;

PROC ENCODE(INT INPUT);
    OUTBUF(4) := BYTE(INPUT LAND BIN 00111111);
    OUTBUF(5) := BYTE((INPUT SRL 6) LAND BIN 00011111);
    OUTBUF(6) := BYTE((INPUT SRL 11) LAND BIN 00011111);
ENDPROC;
```

N-14-3

```
 TITLE MEDTEST
    TESTS THE MICRO-MEDIA ANALOG & DIGITAL INPUTS & OUTPUTS;

% THIS PROGRAM, ALTHOUGH IT DOES NOT USE THE MEDCOM DATA BASE,  %
% USES THE PROCEDURES FROM MEDLNK TO GET AND SEND THE DATA TO/FROM %
% MEDIA. THESE PROCS CALL THE SECURE/RELEASE PROCS IN SECREL, WHICH WE DO
%
% NOT WANT TO HAPPEN HERE, SO THE MODULE SECREL IS NOT INCLUDED AT TASKBUILD
%
% TIME, AND INSTEAD DO-NOTHING SECURE AND RELEASE PROCS ARE PROVIDED INSIDE
%
% THIS MODULE.
%

LET     LF      =       OCT 012;
LET     ENQ     =       OCT 005;
LET     ETX     =       OCT 003;
LET     ESC     =       OCT 033;
LET YES=1;
LET NO=0;
LET RSNGANA = 2;
LET WSNGMED = 12;

MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);

    EXT PROC  (INT,REF MEDCARD)INT SINGLIN,WRITE;
    EXT PROC (INT,INT,INT,REF ARRAY MEDCARD)INT BLOCKIN;
    EXT PROC(REF ARRAY BYTE)TWRT;
    EXT PROC()INT IREAD,OREAD;
    EXT PROC(INT) IWRT,OWRT;
    EXT PROC() TTIO;
    EXT PROC() STARTAST,STOPAST;
    EXT PROC()BYTE ASTCHAR;
    EXT PROC()INT CTLCYET,ASTYET;

SVC DATA RRSIO;
    PROC()BYTE IN;
    PROC(BYTE)OUT;
ENDDATA;

DATA LOCAL;
    INT CHOICE:=0;
    ARRAY(4) INT INPUT :=(0,0,0,0);
    INT DIGDAT := HEX FF00;
    INT NIBLCT := 0;
    INT LOOPCT:=0;
    INT ERRNO;
    MEDCARD MCARD;
ENDDATA;

ENT PROC RRJOB();
  TTIO();
  WHILE CHOICE <1 OR CHOICE >2 DO
    HOME();
    CLEOS();
    TWRT("#LF#MEDIA TESTER");
    TWRT("#LF#============#LF,LF#");
    TWRT("1. DIGITAL#LF#");
    TWRT("2. ANALOGUE#LF#");
    TWRT("->#ENQ#");
    CHOICE:=IREAD();
  REP;
  IF CHOICE=1 THEN
    DIGITAL();
  ELSE
    ANALOGUE();
```

N-15-2

```
  END;
  HOME(); CLEOS();
  OUT(ETX);
ENDPROC;

PROC DIGITAL();
  INT TYPE,SUBTYPE;
  HOME(); CLEOS();
  TWRT("#LF#DIGITAL WORKOUT FOR MEDIA#LF#");
  TWRT("=========================#LF,LF#");
  TWRT("1. DIG INPUTS 1-16 CONNECTED TO DIGOUTS#LF#");
  TWRT("2. DIG INPUTS 17-32 CONNECTED TO DIGOUTS#LF#");
  TWRT("3. DIG INPUTS 1-32 CONNECTED TO DIGOUTS#LF#");
  TWRT("4. MONITOR SWITCHES 1-16#LF#");
  TWRT("5. MONITOR SWITCHES 17-32#LF,LF#");
  TWRT("->#ENQ#");
  CHOICE:=IREAD();
  TYPE:=CHOICE :/ 4;     % 0 : INS TO OUTS ; 1 : MONITOR SWITCHES        %
  SUBTYPE:= TYPE + (CHOICE MOD 4);
  STARTAST();
  IF TYPE=0 THEN  % O/P CONNECTED TO I/P %
    WHILE CTLCYET()=NO DO
      NIBLCT:=(NIBLCT+1)MOD 16;
      DIGDAT := NIBLCT LOR (NIBLCT SLL 4) LOR (NIBLCT SLL 8)
                LOR (NIBLCT SLL 12);

      MCARD.MEDDAT:=DIGDAT;
      MCARD.ADDR:=4;  % DIG OUTPUT %
      MCARD.STAT:=1;
      ERRNO:=WRITE(12,MCARD);
      IF ERRNO#0 THEN
        TWRT("#LF#ERR ");IWRT(ERRNO);TWRT(" ON WRITE TO DIG OUTPUT");OUT(LF);
      ELSE
        IF SUBTYPE=1 THEN % I/P 1-16 %
          GETANDCHECK(0);
        ELSEIF SUBTYPE=2 THEN
          GETANDCHECK(1);
        ELSEIF SUBTYPE=3 THEN
          GETANDCHECK(0);
          GETANDCHECK(1);
        ELSE
          TWRT("#LF#ILLEGAL SUBTYPE #LF#");
        END;
      END;
      LOOPCT:=LOOPCT+1;
      IF LOOPCT MOD 16 = 0 THEN
        TWRT("+#ETX#");
      END;
      IF LOOPCT MOD 1200 = 0 THEN
        TWRT("#LF#");
      END;
    REP;
  ELSE  % SWITCHES %
    WHILE CTLCYET()=NO DO
      MCARD.STAT:=1;
      MCARD.ADDR:=SUBTYPE-1;
      ERRNO:=SINGLIN(1,MCARD);
      IF ERRNO#0 THEN
        TWRT("#LF#ERR ON READ FROM DIGIN ");IWRT(SUBTYPE-1);TWRT("#LF#");
      ELSE
        MCARD.ADDR:=4;
        MCARD.STAT:=1;
```

```
        MCARD.MEDDAT:=NOT(MCARD.MEDDAT);        % BECAUSE DIGINS ARE -VE LOGI
%
        ERRNO:=WRITE(12,MCARD);
        IF ERRNO#0 THEN
          TWRT("#LF#ERROR ON WRITE TO DIGOUT #LF#");
        END;
      END;
    REP;
  END;
  STOPAST();
ENDPROC;

PROC GETANDCHECK(INT N);
  MCARD.ADDR:=N;          %DIGIN  %
  MCARD.STAT:=1;
  MCARD.MEDDAT:=0;
  ERRNO:=SINGLIN(1,MCARD);
  IF ERRNO#0 THEN
    TWRT("#LF#ERR ON READ FROM DIGIN : ERRNO "); IWRT(ERRNO); TWRT("#LF#");
  ELSE
    IF NOT(MCARD.MEDDAT)#DIGDAT THEN    % 'NOT' BECAUSE I/P'S ARE -VE LOGIC%
      TWRT("#LF#DATA INCORRECT : DIGIN NUMBER "); IWRT(N); TWRT("#LF#");
      TWRT("DATA WRITTEN: "); OWRT(DIGDAT);
      TWRT("      DATA READ: "); OWRT(NOT(MCARD.MEDDAT)); TWRT("#LF#");
    END;
  END;
ENDPROC;

PROC ANALOGUE();
  INT TMP2:=0;
  INT TMP:=0;
  HOME(); CLEOS();
  TWRT("#LF#THIS SECTION OF THE PROGRAM READS IN THE SPECIFIED ANALOG ");
  TWRT("#LF#INPUTS AND OUTPUTS THE VALUE READ TO THE SPECIFIED OUTPUTS#LF#");
  FOR I:=1 TO 4 DO
    WHILE INPUT(I)<1 OR INPUT(I)>16 DO
      TWRT("#LF#WHICH INPUT TO OUTPUT "); IWRT(I); TWRT(" #ENQ#");
      INPUT(I):=IREAD();
    REP;
  REP;
  STARTAST();
  WHILE CTLCYET()=NO DO
    FOR I:=1 TO 4 DO
      MCARD.STAT:=1;
      MCARD.ADDR:=INPUT(I);
      MCARD.MEDDAT:=0;
      ERRNO:=SINGLIN(RSNGANA,MCARD);
      IF ERRNO#0 THEN
        TWRT("#LF#ERR ");IWRT(ERRNO); TWRT(" ON READ FROM ANINP ");
        IWRT(I); OUT(LF);
      ELSE
        MCARD.STAT:=1;
        MCARD.ADDR:=OCT 14;
        TMP:=MCARD.MEDDAT LAND OCT 001774;      % MASK ALL XCEPT 8 MSB'S %
        MCARD.MEDDAT:=(TMP SLL 5)LOR (1 SLL (I-1));% SHIFT INTO POSN FOR D/A %
        ERRNO:=WRITE(WSNGMED,MCARD);
        IF ERRNO#0 THEN
          TWRT("#LF#ERR ");IWRT(ERRNO);TWRT(" ON WRITE TO ANINP ");
          IWRT(I);OUT(LF);
        END;
      END;
    REP;
  REP;
```

N-15-3

```
    ERROR(OUTOFRANGE);
  END;
  RETURN (ANINNUM);
ENDPROC;

ENT PROC SETSETPT(INT ANOUTNUM, REAL VALUE);

  % SETS THE SETPOINT OF THE ANALOG INPUT CORRESPONDING TO      %
  % ANALOGUE OUTPUT ANOUTNUM, AS ASSIGNED PREVIOUSLY BY CALLING %
  % SETINPUT.                                                   %

  IF ANOUTNUM > 0 AND ANOUTNUM <= TOTALAO THEN
    TASKINFO(TBUF);
    IF AODESC(ANOUTNUM).UIC = TBUF.TASK17 THEN
      SECMEDCOM();
      AODESC(ANOUTNUM).SETPOINT := VALUE;
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := WRONGUSER;
    END;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  ERROR(ERRNUM);
ENDPROC;

ENT PROC GETSETPT(INT ANOUTNUM)REAL;

  % RETURNS THE SETPOINT FOR THE ANALOG INPUT CORRESPONDING TO  %
  % AN. OUTP ANOUTNUM.                                          %

  REAL VALUE;
  IF ANOUTNUM > 0 AND ANOUTNUM <= TOTALAO THEN
    VALUE := AODESC(ANOUTNUM).SETPOINT;
    ERRNUM := NOERROR;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  ERROR(ERRNUM);
  RETURN(VALUE);
ENDPROC;

PROC ERROR(INT ERRNUM);
  IF ERRNUM#NOERROR THEN
    IF ERP = RRERP THEN
      % might as well give the user a more friendly message     %
      % than the usual RRERP display.                           %
      TWRT("#LF,LF#MEDUSER ERROR -- ");
      IF ERRNUM = WRONGUSER THEN
        TWRT("ATTEMPT TO WRITE TO CHANNEL NOT ATTACHED TO#LF,LF#");
      ELSEIF ERRNUM = BADSWITCH THEN
        TWRT("ADSWITCH OR IOSWITCH NOT 0 OR 1#LF,LF#");
      ELSEIF ERRNUM = OUTOFSCAN THEN
        TWRT("MEDIA INPUT / OUTPUT IS OUT OF SCAN#LF,LF#");
      ELSEIF ERRNUM = OUTOFRANGE THEN
        TWRT("CHANNEL NUMBER OUT OF RANGE#LF,LF#");
      END;
    ELSE
      ERP(700+ERRNUM);
    END;
  END;
```

```
ENDPROC;
```

```
TITLE ABM
 ABORTS TASKS WHICH ACCESS MEDCOM;                                (N-11-1)

% SECURES MEDCOM, ABORTS THE TASK, RELEASES MEDCOM, THEN EXITS  %


LET ETX = 3;
LET ENQ = 5;
LET LF = OCT 12;

MODE R5ONAME(INT R5ON1,R5ON2);
MODE LUNBUF(INT LUNNAME,                    % LUN NAME, E.G. 'TT' OR 'DL'    %
                BYTE LUNDEV,                % DEVICE NO, E.G. 7 FOR TT7: OR %
                                            % 1 FOR DL1:                     %
                LUNDEV1,                    % THE REST IS IRRELEVANT BUMF.   %
                INT LUNCHAR1,
                LUNCHAR2,
                LUNCHAR3,
                LUNSIZE);

EXT PROC() SECMEDCOM,RELMEDCOM; % IN SECREL.RTL %
EXT PROC (INT ) IWRT,RRGEL;
EXT PROC(REF ARRAY BYTE) TWRT;
EXT PROC(REF R5ONAME) R5OREAD,R5OWRT;
EXT PROC () TTIO,GETMCR;

SVC DATA RRTASK;
  R5ONAME MYTASK;
ENDDATA;

SVC DATA RRERRX;
  INT LINENO;
  BYTE UEFLAG,ERRLUN;
  INT R5XDSW;
ENDDATA;

DATA LOCAL;
  LUNBUF LB:=(0,0,0,0,0,0,0);
  R5ONAME NAME;
  REF R5ONAME NAMEP:=NAME;
  INT DS;
  R5ONAME MCRNAME:=(OCT 050712,OCT 131574);     % RADIX-50 FOR 'MCR...' %
ENDDATA;

ENT PROC RRJOB();
  TTIO();

  % ABOM IS THE PRIVILEGED VERSION WHICH CAN ABORT ANY TASK. IT MUST    %
  % NOT BE INSTALLED, SO ONLY PRIVILEGED USERS CAN RUN IT.              %

  % ABM IS THE UNPRIVILEGED VERSION, WHICH SHOULD BE INSTALLED AT HIGH  %
  % PRIORITY AS ...ABM.  IT CAN ABORT PRECISELY THE SAME TASKS THAT     %
  % AN UNPRIVILEGED USER CAN ABORT USING 'ABO'.                         %

  %%%TWRT("TASKNAME #ENQ#");%%%          % THIS LINE FOR ABOM    %
  %%GETMCR();%%                          % THIS LINE FOR ...ABM  %

  R5OREAD(NAMEP);       % GET THE TASK NAME     %
  IF NAME.R5ON1=0 AND NAME.R5ON2=0 THEN
    % USER HAS NOT ENTERED A TASK NAME; USE DEFAULT. (TTnn:)  %
    GETDEFAULTTASKNAME(NAMEP);
  END;
```

```
  IF NAME.R5ON1=MYTASK.R5ON1 AND NAME.R5ON2=MYTASK.R5ON2 THEN
    % USER TRYING TO ABORT THIS TASK : STOP HIM!          %
    TWRT("ABM -- TRIED TO ABORT ABM#ETX#");
  ELSEIF NAME.R5ON1=MCRNAME.R5ON1 AND NAME.R5ON2=MCRNAME.R5ON2 THEN
    % USER TRYING TO ABORT MCR...          %
    TWRT("ABM -- TASK NOT ABORTABLE#ETX#");
  ELSE
    ABORT(NAMEP);
  END;
ENDPROC;

PROC ABORT( REF R5ONAME TASK);

  SECMEDCOM();

  CODE 24,0;
        .MCALL  ABRT$S
        .LIST   MEB
        .GLOBL  $DSW
        MOV     *TASK(R5),R1
        ABRT$S  R1
        MOV     $DSW,*RSXDSW/RRERRX(R0)
        BCC     *OK
*RTL;
                        % IF UNSUCCESSFUL THEN                    %
  DS:= RSXDSW;          %   SAVE DSW FOR LATER                    %
  RELMEDCOM();          %   RELEASE MEDCOM                        %
  CHECKDSW();           %   CHECK WHY NOT SUCCESSFUL              %
  RETURN;               %                                         %
OK:                     % ELSE (* SUCCESSFUL *)                   %
  RELMEDCOM();          %   RELEASE MEDCOM                        %
                        % END                                     %
ENDPROC;

PROC CHECKDSW();

CODE 36,0;
        CMP     *DS/LOCAL,#IE.INS
        BEQ     *NOTINS
        CMP     *DS/LOCAL,#IE.ACT
        BEQ     *NOTACT
        CMP     *DS/LOCAL,#IE.PRI
        BEQ     *NOTPRI
        MOV     *DS/LOCAL,*RSXDSW/RRERRX(R0)
  *RTL;
  TWRT("ABM -- ABORT DIRECTIVE ERROR, DSW=#ETX#");IWRT(RSXDSW);
  RETURN;
NOTINS:
  TWRT("ABM -- TASK NOT IN SYSTEM#ETX#");
  RETURN;
NOTACT:
  TWRT("ABM -- TASK NOT ACTIVE#ETX#");
  RETURN;
NOTPRI:
  TWRT("ABM -- PRIVILEGED COMMAND#ETX#");
ENDPROC;


PROC GETDEFAULTTASKNAME(REF R5ONAME NAME);

% THIS PROC FINDS OUT WHICH TERMINAL THE USER IS SIGNED ONTO.   %
% IT DOES THIS BY A CALL OF GLUN$S MACRO TO GET INFO ABOUT LUN  %
```

N-11-2

```
% NUMBER 1, WHICH WAS ASSIGNED TO TI: DURING TASKBUILD OF ABM.  %
% SEE MODE LUNBUF(....) STATEMENT ABOVE.                        %

  INT TTNO;     % TERMINAL FROM WHICH USER HAS CALLED ABM       %
  INT MSDIGIT,LSDIGIT;  % nn IN TTnn:   %

  LUNINFO(1,LB);        % LUN 1 IS ASSIGNED TO TI:      %
  TTNO := INT(LB.LUNDEV);
  % WE NOW ENCODE TTnn (WHERE nn = TTNO) INTO RADIX 50 FORM.    %
  % nn IS IN OCTAL.                                            %
  LSDIGIT:= TTNO LAND OCT 7;    % LEAST SIGNIFICANT DIGIT       %
  MSDIGIT:= (TTNO SRL 3) LAND OCT 7;
  IF MSDIGIT=0 THEN
    % NEED 'TTn'          %
    MSDIGIT:=LSDIGIT;
    NAME.R50N2:=0;
  ELSE
    NAME.R50N2:=1600*(LSDIGIT + OCT 36);        % 1600 IS (OCT 50)**2   %
  END;
  NAME.R50N1:=OCT 076400 + OCT 001440 + MSDIGIT + OCT 36;
ENDPROC;

PROC LUNINFO(INT LUN,REF LUNBUF BUF);

% EXECUTES GLUN$S DIRECTIVE REQUEST TO GET LUN INFO     %

  CODE 22,0;
        .MCALL  GLUN$S
        .LIST   MEB
        .GLOBL  $DSW
;
        GLUN$S  *LUN(5),*BUF(5)          ; GLUN$S DIRECTIVE
        MOV     $DSW,*RSXDSW/RRERRX(R0)  ; SAVE DSW
        BCC     *OK                      ; DIRECTIVE SUCCESSFUL
  *RTL;

  RRGEL(612);           % DIRECTIVE UNSUCCESSFUL %

OK:
ENDPROC;
```

```
TITLE MEDRMD;                                                    (N-12-1)

% MEDRMD GIVES A RMDEMO-LIKE DISPLAY OF INFORMATION ABOUT MEDIA AND %
% THE STATE OF THE DATA-BASE MEDCOM.                               %

LET LF = OCT 12;
LET ENQ=5;
LET ETX=3;
LET NL = 10;
LET ESC = OCT 33;
LET SP = ' ';
LET CONTROLZ=26;
LET CONTROLC=3;

% numbers of inputs and outputs in MEDCOM :                        %

LET NAO=4;
LET NSIMAO=4;
LET TOTALAO=8;

LET NMULT = 16;
LET NSIMAI = 4;
LET TOTALAI = 20;

LET NDIGICARD = 2;
LET NSIMDIGICARD = 2;
LET TOTALDIGICARD = 4;

LET NDIGOCARD = 1;
LET NSIMDIGOCARD = 1;
LET TOTALDIGOCARD = 2;

LET TIMELINE = 5;       % line on screen where time is displayed      %
LET TIMECOL = 24;
LET DIG = TIMELINE + 4;
LET ANA = DIG + 7;
LET SCANLINE = ANA + 3;
LET SECLINE = SCANLINE + 4; % line where current task is displayed    %
LET SECCOL = 35;

LET YES=1;
LET NO=0;

MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);
MODE AOREC(INT UIC, USERINT, ANIN,REAL SETP);
MODE R5ONAME(INT R5ON1,R5ON2);
MODE IOCL(REF ARRAY BYTE BFR, INT N, DV, PTR, MD, TRM);

EXT PROC (REF R5ONAME) MCOMTASK;        % IN SECREL.RTL %
EXT PROC(INT) DELAY;
EXT PROC () TTIO;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC () BYTE TTIN,HSIN;
EXT PROC (BYTE) HSOUT,TTOUT;
EXT PROC (REF R5ONAME) R5OWRT,R5OREAD;
EXT PROC (REAL)REAL TIMER;
EXT PROC(REAL) RWRTU;
EXT PROC (INT) TIMDAT;
EXT PROC ()INT CTLCYET,ASTYET,NCTLCYET;
EXT PROC (INT)INT WTCCCZ;
EXT PROC ()BYTE ASTCHAR;
EXT PROC(INT)INT WAITAST,WAITCTLC,WAITNCTLC;
```

```
ENT PROC RDMEDIA(INT ADSWITCH,IOSWITCH,CHANNUM) INT;

  % THIS PROC READS ONE OF THE DIGITAL OR ANALOGUE INPUTS OR OUTPUTS %
  %      FROM     %
  % THE DATA BASE AND PUTS THE VALUE READ INTO THE INT VALUE.    %
  % ADSWITCH : =0 FOR ANALOGUE, =1 FOR DIGITAL.                  %
  % IOSWITCH : =0 FOR INPUT, =1 FOR OUTPUT.                      %

  INT VALUE:=0;
  ERRNUM := NOERROR;
  IF ADSWITCH=DIGITAL THEN
    IF IOSWITCH=INPUT THEN
      VALUE:=RDDIGIN(CHANNUM);
    ELSEIF IOSWITCH=OUTPUT THEN
      VALUE:=RDDIGOUT(CHANNUM);
    ELSE
      ERRNUM:=BADSWITCH;
    END;
  ELSEIF ADSWITCH=ANALOG THEN
    IF IOSWITCH=INPUT THEN
      VALUE:=RDANIN(CHANNUM);
    ELSEIF IOSWITCH=OUTPUT THEN
      VALUE:=RDANOUT(CHANNUM);
    ELSE
      ERRNUM:=BADSWITCH;
    END;
  ELSE
    ERRNUM:=BADSWITCH;
  END;
  ERROR(ERRNUM);
  RETURN(VALUE);
ENDPROC;

PROC WRANOUT( INT CHANNUM, MDATA);

  % THIS PROC WRITES THE DATA MDATA TO THE MEDCOM DATA BASE, FROM WHERE %
  % IT IS SENT TO MEDIA BY THE MEDIA UPDATE TASK MEDUPDAT.              %
  % THE OUTPUT CHANNEL MUST HAVE PREVIOUSLY BEEN ATTACHED TO.           %
  % CHANNUM  :  ANALOGUE CHANNEL NUMBER TO WHICH THE DATA       %
  %             IS TO BE SENT.                                  %
  %             ANALOGUE: CHANNUM IN RANGE 1..8                 %
  % MDATA    :  THE 8 LEAST SIGNIFICANT BITS ARE OUTPUT;        %


  IF CHANNUM > 0 AND CHANNUM <= TOTALAO THEN    % CHANNEL IN RANGE      %
    TASKINFO(TBUF);     % GET UIC OF USER PLUS OTHER GARBAGE INTO TBUF  %
    IF AODESC(CHANNUM).UIC = TBUF.TASK17 THEN % CORRECT USER            %

      % BITS 0 TO 3 OF MEDDAT ARE THE CHANNEL SELECT; LEAVE THEM       %
      % ALONE. PUT THE 8 LEAST SIGNIFICANT BITS OF MDATA INTO   BITS   %
      % 7 TO 14 OF MEDDAT.                                             %
      SECMEDCOM();
      IF ANOUTP(CHANNUM).STAT LAND 1 = 1 THEN   % IN SCAN       %
        ANOUTP(CHANNUM).MEDDAT := ((ANOUTP(CHANNUM).MEDDAT) LAND HEX F)
                                  LOR ((MDATA LAND HEX FF) SLL 7);
      ELSE
        ERRNUM := OUTOFSCAN;
      END;
      RELMEDCOM();
    ELSE
      ERRNUM := WRONGUSER;
```

```
    END;
  ELSE
    ERRNUM := OUTOFRANGE;        % CHANNEL NO IS OUT OF RANGE     %
  END;
ENDPROC;

PROC WRDIGOUT (INT CHANNUM,MDATA);
  % THIS PROC WRITES THE DATA MDATA TO THE MEDCOM DATA BASE, FROM WHERE %
  % IT IS SENT TO MEDIA BY THE MEDIA UPDATE TASK MEDUPDAT               %
  % THE OUTPUT CHANNEL CHANNUM MUST HAVE PREVIOUSLY BEEN ATTACHED TO.   %
  % CHANNUM :   THE CHANNEL NUMBER TO WHICH THE DATA IS TO BE SENT.     %
  %             MUST BE IN RANGE 1..32*TOTALDIGOCARD.                   %
  % MDATA   :   INTERPRETED AS ZERO IF MDATA = 0, OTHERWISE ONE.        %

  INT CARDNO, CARDCHAN, MASK;
  IF CHANNUM > 0 AND CHANNUM <= TOTALDIGOCARD*16 THEN
    TASKINFO(TBUF);
    CARDNO := (( CHANNUM - 1) :/ 16) + 1;
    CARDCHAN := (( CHANNUM -1) MOD 16 ) + 1;
    MASK := 1 SLL (16 - CARDCHAN);
    IF DIGUICS(CARDNO,CARDCHAN) = TBUF.TASK17 THEN      % CORRECT USER  %
      SECMEDCOM();
      IF DIGOUT(CARDNO).STAT LAND 1 = 1 THEN    % IN SCAN       %
        IF MDATA = 0 THEN
          % SET BIT CHANNUM (FROM THE LEFT) OF THE DIG O/P TO ZERO      %
          DIGOUT(CARDNO).MEDDAT:= DIGOUT(CARDNO).MEDDAT LAND NOT (MASK);
        ELSE              % REGARD NON-ZERO DATA AS A ONE %
          % SET BIT CHANNUM (FROM THE LEFT) OF THE DIG OUTPUT TO ONE    %
          DIGOUT(CARDNO).MEDDAT := (DIGOUT(CARDNO).MEDDAT) LOR MASK;
        END;
      ELSE
        ERRNUM := OUTOFSCAN;
      END;
      RELMEDCOM();
    ELSE                          % WRONG UIC     %
      ERRNUM := WRONGUSER;
    END;
  ELSE                    % CHANNEL DOES NOT EXIST        %
    ERRNUM := OUTOFRANGE;
  END;
ENDPROC;

PROC RDANIN(INT CHANNUM) INT;
  % THIS PROC READS THE ANALOGUE INPUT NUMBER CHANNUM FROM THE DATA     %
  % BASE AND RETURNS THE VALUE READ.                                    %

  INT VALUE := 0;
  IF CHANNUM > 0 AND CHANNUM <= TOTALAI THEN
    IF ANINP(CHANNUM).STAT LAND 1 = 1 THEN      % IN SCAN       %
      VALUE := (ANINP(CHANNUM).MEDDAT SRL 5) LAND HEX 7FFF ;
      % (BECAUSE BITS 5 TO 14 ARE THE DATA)     %
    ELSE
      ERRNUM := OUTOFSCAN;
    END;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  RETURN (VALUE);
ENDPROC;

PROC RDANOUT ( INT CHANNUM ) INT;
  % THIS PROC RETURNS THE MEDDAT FIELD FROM THE MEDCARD REPRESENTING    %
```

```
  % ANALOGUE OUTPUT NO CHANNUM, I.E. RETRIEVES THE VALUE THAT THE USER  %
  % ATTACHED TO THAT ANALOGUE OUTPUT LAST WROTE TO IT.                   %

  INT VALUE:=0;
  IF CHANNUM > 0 AND CHANNUM <= TOTALAO THEN    % IN RANGE               %
    VALUE := (ANOUTP(CHANNUM).MEDDAT SRL 7)LAND HEX 00FF;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  RETURN ( VALUE );
ENDPROC;

PROC RDDIGIN (INT CHANNUM) INT;
  % THIS PROC READS THE DIGITAL INPUT NUMBER CHANNUM FROM MEDCOM         %
  % AND RETURNS 1 OR 0 ACCORDINGLY.                                      %

  INT VALUE := 0,MASK,CARDNO,CARDCHAN;
  IF CHANNUM > 0 AND CHANNUM <= TOTALDIGICARD*16 THEN
    CARDNO := (( CHANNUM - 1) :/ 16) + 1;
    CARDCHAN := (( CHANNUM -1 ) MOD 16 ) + 1;
    MASK := 1 SLL (16 - CARDCHAN );
    IF DIGINP(CARDNO).STAT LAND 1 = 1 THEN      % IN SCAN       %
      IF DIGINP(CARDNO).MEDDAT LAND MASK = 0 THEN
        % CHANNEL CHANNUM CONTAINS A ZERO(NEGATIVE LOGIC) SO RETURN A ONE
%
        VALUE := 1;
      ELSE
        VALUE := 0;
      END;
    ELSE
      ERRNUM := OUTOFSCAN;
    END;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;

  RETURN(VALUE);

ENDPROC;

PROC RDDIGOUT ( INT CHANNUM) INT;
  % THIS PROC RETURNS THE VALUE (0 OR 1) MOST RECENTLY WRITTEN TO DIGITAL %
  % OUTPUT NUMBER CHANNUM.                                                 %

  INT VALUE:=0,CARDNO,CARDCHAN,MASK;
  IF CHANNUM > 0 AND CHANNUM <= 16*TOTALDIGOCARD THEN   % IN RANGE      %
    CARDNO := (( CHANNUM - 1) :/ 16) + 1;
    CARDCHAN := (( CHANNUM - 1) MOD 16) +1;
    MASK := 1 SLL (16 - CARDCHAN);

    VALUE := IF DIGOUT(CARDNO).MEDDAT LAND MASK = 0 THEN
               0
             ELSE
               1
             END;

  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  RETURN ( VALUE );
ENDPROC;

ENT PROC ATTACHED (INT ADSWITCH,REF ARRAY INT OUTARRAY);
```

```
  % THIS PROC EXAMINES THE DIGITAL OR ANALOGUE OUTPUT MEDCARDS IN        %
  % MEDCOM AND PUTS INTO THE ARRAY OUTARRAY THE CHANNEL NUMBERS OF       %
  % THOSE TO WHICH THE USER IS ATTACHED.                                 %
  % E.G. IF ADSWITCH=DIGITAL AND THE USER IS ATTACHED TO DIGITAL         %
  % OUTPUTS 1,5 AND 7 THEN AFTER CALLING THIS PROC, OUTARRAY(1) WILL     %
  % BE 1, OUTARRAY(2) WILL BE 5 AND OUTARRAY(3) WILL BE 7. THE REST      %
  % OF THE ARRAY WILL BE CLEARED TO ZEROS.                               %

  INT ARRAYCOUNT:=0;
  ERRNUM:=NOERROR;

  TASKINFO(TBUF);        % GET INFO ABOUT USER (INCLUDING UIC)   %

  FOR I:=1 TO LENGTH OUTARRAY DO
    OUTARRAY(I) := 0;
  REP;

  IF ADSWITCH = ANALOG THEN
    FOR I:=1 TO TOTALAO DO
      IF AODESC(I).UIC = TBUF.TASK17 THEN       % USER IS ATTACHED      %
        ARRAYCOUNT := ARRAYCOUNT + 1;    % POINT TO NEXT INT IN ARRAY    %
        OUTARRAY(ARRAYCOUNT) := I;
      END;
    REP;
  ELSEIF ADSWITCH = DIGITAL THEN
    FOR CARDNUM := 1 TO TOTALDIGOCARD DO
      FOR I := 1 TO 16 DO
        IF DIGUICS (CARDNUM,I) = TBUF.TASKUIC THEN
          ARRAYCOUNT := ARRAYCOUNT + 1;
          OUTARRAY(ARRAYCOUNT) := (CARDNUM - 1)*16 + I;
        END;
      REP;
    REP;
  ELSE          % ADSWITCH NOT VALID    %
    ERRNUM := BADSWITCH;
  END;
  ERROR(ERRNUM);
ENDPROC;

ENT PROC RDCOMMINT (INT CHANNUM) INT;
  % THIS PROC RETURNS THE VALUE OF AODESC.USERINT CORRESPONDING TO       %
  % ANALOGUE OUTPUT NUMBER CHANNUM.  THIS INTEGER CAN BE USED BY THE     %
  % USER FOR INTERTASK COMMUNICATION.                                    %

  INT RET :=0;
  IF CHANNUM > 0 AND CHANNUM <= TOTALAO THEN
    RET := AODESC (CHANNUM).USERINT ;
  ELSE
    ERROR ( OUTOFRANGE );
  END;
  RETURN (RET);
ENDPROC;

ENT PROC WRCOMMINT (INT CHANNUM, VALUE );
  % THIS PROC WRITES THE CONTENTS OF VALUE INTO THE USERINT FIELD        %
  % OF THE AOREC CORRESPONDING TO ANALOGUE OUTPUT NUMBER CHANNUM.        %
  % THIS INTEGER IS INTENDED TO BE USED FOR INTERTASK COMMUNICATION BY   %
  % USER TASKS.                                                          %

  IF CHANNUM >0 AND CHANNUM <= TOTALAO THEN
    TASKINFO (TBUF);
```

```
    IF AODESC (CHANNUM).UIC = TBUF.TASK17 THEN  % CORRECT USER  %
      SECMEDCOM();
      AODESC(CHANNUM).USERINT := VALUE;
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := WRONGUSER;
    END;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  ERROR (ERRNUM);
ENDPROC;

ENT PROC READMEDCARD(INT ADSWITCH, IOSWITCH, CARDNUM, REF MEDCARD CARD);

  % TAKES THE CONTENTS OF THE REQUIRED MEDCARD FROM THE MEDIA DATA BASE %
  % AND PUTS IT INTO "CARD".                                           %
  % ADSWITCH : CHOOSES ANALOG(0) OR DIGITAL(1).                        %
  % IOSWITCH : CHOOSES INPUT(0) OR OUTPUT(1).                          %
  % CARDNUM :  CHOOSES THE CARD NUMBER WITHIN THE CARD TYPE SPECIFIED  %
  %     BY ADSWITCH AND IOSWITCH.                                      %
  % CARD     : VARIABLE OF TYPE MEDCARD INTO WHICH THE INFO IS PUT.    %

  IF ADSWITCH = ANALOG AND IOSWITCH = INPUT THEN
    IF CARDNUM > 0 AND CARDNUM <= TOTALAI THEN
      SECMEDCOM();
      COPYCARD(ANINP(CARDNUM),CARD);    % PUT THE MEDCARD INTO CARD     %
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := OUTOFRANGE;
    END;
  ELSEIF ADSWITCH=DIGITAL AND IOSWITCH = INPUT THEN
    IF CARDNUM > 0 AND CARDNUM <= TOTALDIGICARD THEN
      SECMEDCOM();
      COPYCARD(DIGINP(CARDNUM),CARD);
      RELMEDCOM();
      CARD.MEDDAT:=NOT(CARD.MEDDAT);    % CONVERT DIGIN DATA TO POS LOGIC %
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := OUTOFRANGE;
    END;
  ELSEIF ADSWITCH = DIGITAL AND IOSWITCH = OUTPUT THEN
    IF CARDNUM > 0 AND CARDNUM <= TOTALDIGOCARD THEN
      SECMEDCOM();
      COPYCARD(DIGOUT(CARDNUM),CARD);
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := OUTOFRANGE;
    END;
  ELSEIF ADSWITCH = ANALOG AND IOSWITCH = OUTPUT THEN
    IF CARDNUM > 0 AND CARDNUM <= TOTALAO THEN
      SECMEDCOM();
      COPYCARD(ANOUTP(CARDNUM),CARD);
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := OUTOFRANGE;
    END;
  ELSE
```

```
      RELMEDCOM();
      ERRNUM := NOERROR;
    ELSE
      ERRNUM := WRONGUSER;
    END;
  ELSE
    ERRNUM := OUTOFRANGE;
  END;
  ERROR(ERRNUM);
ENDPROC;

ENT PROC GETANINP(INT ANOUTNUM)INT;
  % RETURNS THE CHANNEL NO OF THE ANALOGUE INPUT BEING CONTROLLED BY    %
  % ANALOGUE OUTPUT ANOUTNUM, AS SET UP BY A PREVIOUS CALL TO SETANINP. %

  INT ANINNUM:=0;
  IF ANOUTNUM > 0 AND ANOUTNUM <= TOTALAO THEN
    ANINNUM := AODESC(ANOUTNUM).ANIN;
  ELSE
```

N-10-7

```
EXT PROC() STARTAST,STOPAST;
EXT PROC (INT,INT) GOTOLC;
EXT PROC ()HOME,CLEOS,CLEOL;

SVC DATA RRSIO; PROC()BYTE IN; PROC(BYTE)OUT ENDDATA;
SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRCHAN; REF IOCL INCL, OUTCL ENDDATA;

% The following 2 bricks are MEDCOM :          %

EXT DATA OUTAREA;
  ARRAY(TOTALAO) MEDCARD ANOUTP;
  ARRAY(TOTALDIGOCARD) MEDCARD DIGOUT;
  ARRAY(TOTALAO) AOREC AODESC;
  ARRAY(TOTALDIGOCARD,16) INT DIGUICS;
ENDDATA;

EXT DATA INAREA;
  ARRAY(TOTALAI) MEDCARD ANINP;
  ARRAY(TOTALDIGICARD) MEDCARD DIGINP;
  ARRAY(2) MEDCARD DIGCHAN;
  MEDCARD MEDSTAT;
ENDDATA;

DATA IOLOCAL;
  ARRAY(132)BYTE IBUF,OBUF:=(NL,SP(131));
  IOCL ICL := ( IBUF,0,1,0,1,0);        % HAVE SET MD TO 1 FOR BINARY   %
  IOCL OCL := ( OBUF,0,1,1,1,0);        % DITTO          %
ENDDATA;

DATA LOCAL;
  ARRAY(16)INT DUIC;
  ARRAY(4)INT AUIC;
  R50NAME CURTASK;
  ARRAY(4) INT TAB := (1,20,40,60);
  ARRAY(5) INT TAB2:= (1,16,31,46,61);
  ARRAY(32) REF ARRAY BYTE NUMBERS := (" 1. "," 2. "," 3. "," 4. ",
     " 5. "," 6. "," 7. "," 8. "," 9. ","10. ","11. ","12. ","13. ",
     "14. ","15. ","16. ","17. ","18. ","19. ","20. ","21. ","22. ",
     "23. ","24. ","25. ","26. ","27. ","28. ","29. ","30. ","31. ","32. ");
  ARRAY(9) BYTE UBUF;
  ARRAY(5) INT INSCAN;
  ARRAY(6) REF ARRAY BYTE SCANBUF := ("Digin1 ","Digin2 ","Digout ",
                                      "Anin ","Anout ","*OUT OF SCAN*");
  REF ARRAY BYTE PARTID := "";
  INT CHAR,DIGICARDMIN,DIGOCARD,AOMIN,AIMIN;
ENDDATA;

ENT PROC RRJOB();

  INT UICTMP,EXIT:=NO;
  BYTE INCHAR;

  INITIALISEIO();
  STARTAST();
  INITFORMEDIA();
  WHILE EXIT=NO DO
    INITIALISEDATA();
    INITIALISESCREEN();
    LOOP();
    CHAR := ASTCHAR();
    IF CHAR = CONTROLC OR CHAR = CONTROLZ THEN
```

```
      % user wants to exit from MEDRMD  %
      EXIT := YES;
    ELSEIF CHAR = 'S' OR CHAR = 's' THEN
      % user wants simulation info      %
      INITFORSIM();
    ELSE
      % user wants real Media info      %
      INITFORMEDIA();
    END;
  REP;
  STOPAST();
  HOME();CLEOS();FORCEBUFFEROUTPUT();
ENDPROC;

PROC INITFORMEDIA();
  PARTID:=" ( Real Media system )";
  AIMIN :=1;
  AOMIN :=1;
  DIGICARDMIN :=1;
  DIGOCARD :=1;
ENDPROC;

PROC INITFORSIM();
  PARTID := " ( Simulation part )";
  AIMIN := 17;
  AOMIN := 5;
  DIGICARDMIN := 3;
  DIGOCARD := 2;
ENDPROC;



PROC LOOP();

  INT UICTMP;
  INT STOP:=NO;
  WHILE STOP=NO DO
    PRINTTIME();

    FOR I:=1 TO 16 DO
      UICTMP:=DIGUICS(DIGOCARD,I);
      IF DUIC(I)#UICTMP THEN
        DUIC(I):=UICTMP;
        GTODOUT(I);
        UICWRT(UICTMP);
      END;
    REP;

    FOR I:=1 TO 4 DO
      UICTMP:=AODESC(I-1+AOMIN).UIC;
      IF AUIC(I)#UICTMP THEN
        AUIC(I):=UICTMP;
        GTOAOUT(I);
        UICWRT(UICTMP);
      END;
    REP;

    GOTOLC(SCANLINE+2,1); CLEOL();
    SCANWRT();

    MCOMTASK(CURTASK);
    GOTOLC(SECLINE,SECCOL);
```

```
    CLEOL();
    IF CURTASK.R50N1#0 THEN
      R50WRT(CURTASK);
    END;

    HOME();
    FORCEBUFFEROUTPUT();

    IF WAITAST(50)=YES THEN
      STOP:=YES;
    END;
  REP;
ENDPROC;

PROC INITIALISEDATA();

  FOR I:=1 TO 16 DO
    DUIC(I):= DIGUICS(DIGOCARD,I);
  REP;
  FOR I:=1 TO 4 DO
    AUIC(I):= AODESC(I-1+AOMIN).UIC;
  REP;
  MCOMTASK(CURTASK);
ENDPROC;

PROC INITIALISESCREEN();

  HOME();
  CLEOS();

  GOTOLC(1,11); TWRT("Dynamic display of usage of MEDCOM data base");
  GOTOLC(2,11); TWRT("==========================================");
  GOTOLC(3,22); TWRT(PARTID);

  PRINTTIME();

  GOTOLC(DIG-2,1); TWRT("Digital Outputs :");

  FOR I:= 1 TO 4 DO
    FOR J:=1 TO 4 DO
      GOTOLC(DIG+J-1,TAB(I));
      TWRT(NUMBERS((DIGOCARD - 1)*16 + 4*(I-1) + J));
      UICWRT(DUIC(4*(I-1) + J));
    REP;
  REP;

  GOTOLC(ANA-2,1); TWRT("Analog outputs :");
  FOR I:=1 TO 4 DO
    GOTOLC(ANA,TAB(I));
    TWRT(NUMBERS(AOMIN - 1 + I)); UICWRT(AUIC(I));
  REP;

  GOTOLC(SCANLINE,1); TWRT("Seconds since last update : ");
  FOR I:=1 TO 5 DO
    GOTOLC(SCANLINE+1,TAB2(I)); TWRT(SCANBUF(I));
  REP;

  GOTOLC(SECLINE,1); TWRT("Task currently securing MEDCOM :");
ENDPROC;


PROC INITIALISEIO();
```

N-12-4

```
% BASED ON TTIO() BUT  USE HSIN,HSOUT FOR BINARY DATA XFER      %
  IN:=HSIN;
  OUT:=HSOUT;
  INCL:=ICL;
  OUTCL:=OCL;
ENDPROC;

PROC PRINTTIME();
  GOTOLC(TIMELINE,TIMECOL);
  TIMDAT(-1);
ENDPROC;

PROC UICWRT(INT UIC);
 IF UIC # 0 THEN
  UBUF(9):=']';
  UBUF(8):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(7):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(6):=BYTE((UIC LAND 3) + '0');
  UIC:=UIC SRL 2;
  UBUF(5):=',';
  UBUF(4):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(3):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(2):=BYTE((UIC LAND 3) + '0');
  UBUF(1):='[';
  FOR I:=1 TO 9 DO
    OUT(UBUF(I));
  REP;
 ELSE
  FOR I:=1 TO 9 DO OUT(' '); REP;
 END;
ENDPROC;

PROC GTODOUT(INT N);
  GOTOLC(DIG + ((N-1) MOD 4), 4 + TAB((N-1) :/ 4 + 1));
ENDPROC;

PROC GTOAOUT(INT N);
  GOTOLC(ANA , 4 + TAB(N));
ENDPROC;

PROC FORCEBUFFEROUTPUT();
  TTOUT(ETX);
ENDPROC;

PROC SCANWRT();

 REAL SCANT;

  FOR I:=1 TO 2 DO
    GOTOLC(SCANLINE+2,TAB2(I));
    IF DIGINP(I-1+DIGICARDMIN).STAT LAND 1 = 1 THEN
      SCANT:=DIGINP(I-1+DIGICARDMIN).SCANTIME;
      RWRTU(TIMER(SCANT));
    ELSE
      TWRT(SCANBUF(6));
    END;
```

N-12-5

```
  REP;

  GOTOLC(SCANLINE+2,TAB2(3));
  IF DIGOUT(DIGOCARD).STAT LAND 1 = 1 THEN
    SCANT:=DIGOUT(DIGOCARD).SCANTIME;
    RWRTU(TIMER(SCANT));
  ELSE
    TWRT(SCANBUF(6));
  END;

  GOTOLC(SCANLINE+2,TAB2(4));
  IF ANINP(AIMIN).STAT LAND 1 = 1 THEN
    SCANT:=ANINP(AIMIN).SCANTIME;
    RWRTU(TIMER(SCANT));
  ELSE
    TWRT(SCANBUF(6));
  END;

  GOTOLC(SCANLINE+2,TAB2(5));
  IF ANOUTP(AOMIN).STAT LAND 1 = 1 THEN
    SCANT:=ANOUTP(AOMIN).SCANTIME;
    RWRTU(TIMER(SCANT));
  ELSE
    TWRT(SCANBUF(6));
  END;
ENDPROC;
```

```
  REP;
  IF FOUND = YES THEN
    TWRT("#LF#ATTACHED TO DIGITAL OUTPUT NO "); IWRT(CHAN);
  ELSE
    TWRT("#LF#COULD NOT ATTACH -- NO FREE CHANNELS#LF#");
  END;
ENDPROC;

PROC ATTACHDIGOUT();
  INT CARDNO,CARDCHAN;
  INT DONE:=NO,CHAN:=0;
  WHILE CHAN < 1 OR CHAN > 16*TOTALDIGOCARD DO
    TWRT("#LF#DIGITAL OUTPUT NUMBER #ENQ#");
    CHAN:=IREAD();
  REP;
  CARDNO:= (( CHAN -1) :/ 16) +1;
  CARDCHAN := (( CHAN - 1) MOD 16) + 1;
  IF DIGUICS(CARDNO,CARDCHAN)=0 AND DIGOUT(CARDNO).STAT LAND 1 = 1 THEN
    SECMEDCOM();
    IF DIGUICS(CARDNO,CARDCHAN) = 0 THEN
      DIGUICS(CARDNO,CARDCHAN):=TBUF.TASK17;
      DONE :=YES;
    END;
    RELMEDCOM();
  END;
  IF DONE=YES THEN
    TWRT("#LF#ATTACHMENT COMPLETE");
  ELSE
    TWRT("#LF#COULD NOT ATTACH -- OUT OF SCAN OR ALREADY ATTACHED#LF#");
  END;
ENDPROC;

PROC FINDANDATTACHANOUT(INT CHOICE);
  INT FOUND,I,CHAN,LOWCHAN,HIGHCHAN;
  IF CHOICE=4 THEN
    % REAL MEDIA %
    LOWCHAN := 1;
    HIGHCHAN := NAO;
  ELSE  % SIMULATION %
    LOWCHAN := NAO + 1;
    HIGHCHAN := TOTALAO;
  END;

  I := LOWCHAN;
  FOUND:=NO;
  WHILE FOUND=NO AND I <= HIGHCHAN DO
    IF AODESC(I).UIC=0 AND ANOUTP(I).STAT LAND 1 = 1 THEN
      SECMEDCOM();
      IF AODESC(I).UIC=0 THEN
        AODESC(I).UIC:=TBUF.TASK17;
        FOUND:=YES;
        CHAN:=I;
      END;
      RELMEDCOM();
    END;
    I := I + 1;
  REP;
  IF FOUND=YES THEN
    TWRT("#LF#ATTACHED TO ANALOG OUTPUT NUMBER ");IWRT(CHAN);TWRT("#LF#");
  ELSE
    TWRT("#LF#COULD NOT ATTACH -- NO FREE CHANNELS#LF#");
  END;
```

```
ENDPROC;

PROC ATTACHANOUT();
  INT CHAN:=0,DONE:=NO;
  WHILE CHAN < 1 OR CHAN > TOTALAO DO
    TWRT("#LF#ANALOG OUTPUT NUMBER #ENQ#");
    CHAN:=IREAD();
  REP;
  IF AODESC(CHAN).UIC=0 AND ANOUTP(CHAN).STAT LAND 1 = 1 THEN
    SECMEDCOM();
    IF AODESC(CHAN).UIC=0 THEN
      AODESC(CHAN).UIC:=TBUF.TASK17;
      DONE:=YES;
    END;
    RELMEDCOM();
  END;
  IF DONE=YES THEN
    TWRT("#LF#ATTACHMENT COMPLETE");
  ELSE
    TWRT("#LF#COULD NOT ATTACH -- OUT OF SCAN OR ALREADY ATTACHED");
  END;
ENDPROC;

PROC DETACH();
  INT CHAN,CHOICE,BUSY;
  BUSY:=YES;
  WHILE BUSY=YES AND QUIT=NO DO
    HOME();CLEOL();
    TWRT("DETACH MENU#LF#"); CLEOL(); OUT(LF); CLEOL();
    TWRT("1. Detach from digital output#LF#"); CLEOL();
    TWRT("2. Detach from analogue output#LF#"); CLEOL();
    TWRT("3. Detach all outputs#LF#"); CLEOL();
    TWRT("4. Return to main menu#LF#"); CLEOL();
    TWRT("5. Quit#LF#"); CLEOL(); OUT(LF); CLEOL(); OUT(ENQ);
    CHOICE:=IREAD();
    CLEOS();
    IF CHOICE=1 THEN    % DIGITAL       %
      DETACHDIG();
    ELSEIF CHOICE=2 THEN        % ANALOGUE      %
      DETACHANA();
    ELSEIF CHOICE=3 THEN
      DETACHALL();
    ELSEIF CHOICE=4 THEN
      BUSY := NO;
    ELSEIF CHOICE=5 THEN
      QUIT := YES;
    END;
  REP;
ENDPROC;

PROC DETACHDIG();
  INT CARDNO,CARDCHAN,CHAN:=0;
  WHILE CHAN < 1 OR CHAN > 16*TOTALDIGOCARD DO
    TWRT("#LF#DIGITAL OUTPUT NUMBER #ENQ#");
    CHAN:=IREAD();
  REP;
  CARDNO := ((CHAN - 1) :/ 16) + 1;
  CARDCHAN := ((CHAN - 1) MOD 16) + 1;
  IF MANAGER=YES THEN
    IF DIGUICS(CARDNO,CARDCHAN)=0 THEN
      TWRT("#LF#COULD NOT DETACH -- NO USER ATTACHED");
    ELSE
```

```
      SECMEDCOM();
      DIGUICS(CARDNO,CARDCHAN):=0;
      RELMEDCOM();
      TWRT("#LF#DETACHMENT COMPLETE");
    END;
  ELSE  % NOT MANAGER   %
    IF DIGUICS(CARDNO,CARDCHAN)=TBUF.TASK17 THEN
      SECMEDCOM();
      DIGUICS(CARDNO,CARDCHAN):=0;      % DETACH        %
      RELMEDCOM();
      TWRT("#LF#DETACHMENT COMPLETE");
    ELSE
      TWRT("#LF#COULD NOT DETACH -- USER NOT ATTACHED#LF#");
    END;
  END;
ENDPROC;

PROC DETACHANA();
  INT CHAN:=0;
  WHILE CHAN < 1 OR CHAN > TOTALAO DO
    TWRT("#LF#ANALOG OUTPUT NUMBER #ENQ#");
    CHAN:=IREAD();
  REP;
  IF MANAGER=YES THEN
    IF AODESC(CHAN).UIC = 0 THEN
      TWRT("#LF#COULD NOT DETACH -- NO USER ATTACHED");
    ELSE
      SECMEDCOM();
      AODESC(CHAN).UIC:=0;
      AODESC(CHAN).USERINT:=0;
      AODESC(CHAN).ANIN:=0;
      AODESC(CHAN).SETPOINT:=0.0;
      RELMEDCOM();
      TWRT("#LF#DETACHMENT COMPLETE");
    END;
  ELSE
    IF AODESC(CHAN).UIC=TBUF.TASK17 THEN
      SECMEDCOM();
      AODESC(CHAN).UIC:=0;
      AODESC(CHAN).USERINT:=0;
      AODESC(CHAN).ANIN:=0;
      AODESC(CHAN).SETPOINT:=0.0;
      RELMEDCOM();
      TWRT("#LF#DETACHMENT COMPLETE");
    ELSE
      TWRT("#LF#COULD NOT DETACH -- USER NOT ATTACHED#LF#");
    END;
  END;
ENDPROC;

PROC DETACHALL();
  IF  MANAGER=YES THEN  % MANAGER       %
    SECMEDCOM();
    FOR I:=1 TO TOTALAO DO
      AODESC(I).UIC:=0;
      AODESC(I).USERINT :=0;
      AODESC(I).ANIN:=0;
      AODESC(I).SETPOINT:=0.0;
    REP;
    FOR CARDCOUNT := 1 TO TOTALDIGOCARD DO
      FOR I:=1 TO 16 DO
        DIGUICS(CARDCOUNT,I):=0;
```

```
      REP;
    REP;
    RELMEDCOM();
    TWRT("#LF#FUNCTION COMPLETE -- ALL USERS OF OUTPUTS DETACHED#LF#");
  ELSE              % NON-MANAGER USER       %
    FOR I:=1 TO TOTALAO DO
      IF AODESC(I).UIC=TBUF.TASK17 THEN
        SECMEDCOM();
        AODESC(I).UIC:=0;
        AODESC(I).USERINT := 0;
        AODESC(I).ANIN:=0;
        AODESC(I).SETPOINT:=0.0;
        RELMEDCOM();
      END;
    REP;
    FOR CARDCOUNT := 1 TO TOTALDIGOCARD DO
      FOR I:=1 TO 16 DO
        IF DIGUICS(CARDCOUNT,I)=TBUF.TASK17 THEN        % THIS USER'S   %
          SECMEDCOM();
          DIGUICS(CARDCOUNT,I):=0;
          RELMEDCOM();
        END;
      REP;
    REP;
    TWRT("#LF#FUNCTION COMPLETE -- ALL ");UICWRT(TBUF.TASK17);
    TWRT(" OUTPUTS DETACHED#LF#");
  END;
ENDPROC;

PROC UICWRT(INT UIC);
  UBUF(9):=']';
  UBUF(8):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(7):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(6):=BYTE((UIC LAND 3) + '0');
  UIC:=UIC SRL 2;
  UBUF(5):=',';
  UBUF(4):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(3):=BYTE((UIC LAND 7) + '0');
  UIC:=UIC SRL 3;
  UBUF(2):=BYTE((UIC LAND 3) + '0');
  UBUF(1):='[';
  TWRT(UBUF);
ENDPROC;
```

```
PROC ATTACH();
  INT BUSY:=YES;
  INT CHOICE,CHAN,FOUND,DONE;
  WHILE QUIT=NO AND BUSY=YES DO
    HOME();
    CLEOL(); TWRT("ATTACH MENU#LF#");
    CLEOL(); OUT(LF);
    CLEOL(); TWRT("1. Any digital output (real Media)#LF#");
    CLEOL(); TWRT("2. Any digital output (simulation)#LF#");
    CLEOL(); TWRT("3. Specific digital output #LF,LF#");CLEOL();
    CLEOL(); TWRT("4. Any analogue output (real Media)#LF#");
    CLEOL(); TWRT("5. Any analogue output (simulation)#LF#");
    CLEOL(); TWRT("6. Specific analogue output#LF,LF#");CLEOL();
    CLEOL(); TWRT("7. Return to main menu#LF#");
    CLEOL(); TWRT("8. Quit#LF#");
    CLEOL(); OUT(LF); CLEOL(); OUT(ENQ);
    CHOICE:=IREAD();
    CLEOS();
    IF CHOICE=1 OR CHOICE=2 THEN
      FINDANDATTACHDIGOUT(CHOICE);
    ELSEIF CHOICE=3 THEN
      ATTACHDIGOUT();
    ELSEIF CHOICE=4 OR CHOICE=5 THEN
      FINDANDATTACHANOUT(CHOICE);
    ELSEIF CHOICE=6 THEN
      ATTACHANOUT();
    ELSEIF CHOICE=7 THEN
      BUSY:=NO;
    ELSEIF CHOICE=8 THEN
      QUIT:=YES;
    END;
  REP;
ENDPROC;

PROC FINDANDATTACHDIGOUT(INT CHOICE);
  INT CARDNO,I,CHAN,FOUND,LOWCARD,HIGHCARD;
  IF CHOICE=1 THEN
    LOWCARD :=1;
    HIGHCARD := NDIGOCARD;
  ELSE
    LOWCARD := NDIGOCARD+1;
    HIGHCARD := TOTALDIGOCARD;
  END;

  CARDNO:=LOWCARD;
  WHILE FOUND = NO AND CARDNO <= HIGHCARD DO
    I:=1;
    WHILE FOUND = NO AND I <= 16 DO
      IF DIGUICS (CARDNO,I) = 0 AND DIGOUT(CARDNO).STAT LAND 1 = 1 THEN
        % NO-ONE ATTACHED AND CARD IS IN SCAN   %
        SECMEDCOM();
        IF DIGUICS(CARDNO,I) = 0 THEN
          % STILL NO-ONE ATTACHED       %
          DIGUICS(CARDNO,I) := TBUF.TASK17;
          FOUND := YES;
          CHAN := (CARDNO-1)*16 + I;
        END;
        RELMEDCOM();
      END;
      I:=I+1;
    REP;
    CARDNO := CARDNO + 1;
```

```
ENT PROC RELMEDCOM();

  RELEASE(MEDCOMEF);
ENDPROC;

ENT PROC MCOMINIT();

% THIS PROC IS INTENDED TO BE CALLED, ONCE, AT THE START OF THE MEDIA   %
% UPDATE TASK.                                                          %
% INITIALISES MEDCOM FOR SECURE AND RELEASE, BY SETTING FLAG MEDCOMEF.  %
% THIS IS NECESSARY BECAUSE THE SYSTEM WAKES UP WITH ALL FALGS ZERO,    %
% WHICH REPRESENTS FACILITIES SECURED (SEE COMMENTS TO SECURE PROC).    %

  SET (MEDCOMEF);
ENDPROC;

ENT PROC MCOMTASK(REF R50NAME TASK);

% THIS PROC RETURNS TO THE CALLING PROGRAM THE RADIX-50 NAME OF THE     %
% TASK WHICH IS CURRENTLY SECURING MEDCOM.                              %

  TASK.R50N1:=GLFACS(MEDCOMEF-32).R50N1;
  TASK.R50N2:=GLFACS(MEDCOMEF-32).R50N2;
ENDPROC;

ENT PROC FREEMEDCOM();

% THIS PROC IS INTENDED FOR A SYSTEM MANAGER TO FREE MEDCOM SHOULD IT   %
% SOMEHOW BECOME SECURED, NOT RELEASED AND HENCE 'HANG'.                %

    FORCEDRELEASE(MEDCOMEF);
ENDPROC;

ENT PROC TASKINFO(REF TASKBUF TB);

% AN EQUIVALENT TO RSXGTS TO DO THE GTSK$S MACRO DOES NOT EXIST IN      %
% MTSLIB, SO HERE IT IS.                                                %

  CODE 18,0;
        .MCALL  GTSK$S
        .LIST   MEB
        .GLOBL  $DSW
        GTSK$S  *TB(5)
        MOV     $DSW,*RSXDSW/RRERRX(R0)
        BCC     *OK1
  *RTL;

  RRGEL(14);    % DIRECTIVE UNSUCCESSFUL        %
OK1:
ENDPROC;

PROC SECURE(INT FA);

% THE 2 PROCS HERE ARE JUST LIKE THE MTSLIB SECURE & RELEASE, EXCEPT    %
% THAT THEY USE THE GLOBAL EVENT FLAGS 33 TO 56 INSTEAD OF MTSLIB'S     %
% GROUP GLOBAL FLAGS. SINCE THE SYSTEM WAKES UP WITH THE FLAGS ZERO,    %
% AND ZERO MEANS SECURED, IT IS NECESSARY TO EXPLICITLY SET             %
% THE FLAGS BEFORE ANY TASKS CALLING SECURE ARE RUN (EG AT SYSTEM       %
% STARTUP).                                                             %
% NOTE: THE METHOD USED IN MTSLIB SECURE - TO ALLOW THE CALLER TO GRAB  %
% THE FACILITY IF THE R50N1 OF THE SECURING TASK IS ZERO (EVEN IF THE   %
```

```
% FLAG IS STILL CLEAR !) DOES NOT WORK.   IT IS RECOMMENDED THAT THE     %
% MTSLIB PROC BE CHANGED TO RESEMBLE THIS ONE, SINCE AT PRESENT, USE     %
% OF IT CAUSES MULTIPLE ERP(214) & (215)'S IF 2 TASKS ARE COMPETING FOR %
% THE FACILITY, AND ERP(214) FOLLOWED BY RRGEL(15) IF MORE THAN 2 TASKS %
% ARE COMPETING.                                                        %

REF R5ONAME TASK;

  RSXDSW:=FA;
  IF FA < 33 OR FA > 56 THEN
    RRGEL(14);
  END;

  TASK:=GLFACS(FA-32);
  IF TASK.R5ON1=MYTASK.R5ON1 AND TASK.R5ON2=MYTASK.R5ON2 THEN
    RRGEL(16);  % ALREADY SECURED       %
  END;

  CODE 20,0;
        .MCALL   CLEF$S,WTSE$S
        .LIST    MEB
        .GLOBL   $DSW
AGAIN:  CLEF$S   *FA(R5)
        BCS      DERR              ; DIRECTIVE ERROR
        CMP      #IS.SET,$DSW      ;  WAS IT SET ?
        BEQ      *GOTIT
  *RTL;
  CODE 14,0;
        WTSE$S   *FA(R5)
        BCS      DERR
        BR       AGAIN
  *RTL;
GOTIT:
  TASK.R5ON1:=MYTASK.R5ON1;
  TASK.R5ON2:=MYTASK.R5ON2;      % GRAB THE FACILITY     %
ENDPROC;

PROC RELEASE(INT FA);
  REF R5ONAME TASK;
  RSXDSW:=FA;
  IF FA < 33 OR FA > 56 THEN
    RRGEL(14);
  END;
  TASK:=GLFACS(FA-32);
  IF TASK.R5ON1 # MYTASK.R5ON1 OR TASK.R5ON2 # MYTASK.R5ON2 THEN
    RRGEL(15);  % FACILITY NOT SECURED  %
  END;

  TASK.R5ON1 := TASK.R5ON2 := 0;

  CODE 20,0;
        .MCALL   SETF$S,DECL$S
        SETF$S   *FA(R5)
        BCS      DERR
        CMP      #IS.SET,$DSW
        BNE      *OK
  *RTL;
  ERP(215);      % SECURED,BUT FLAG WAS SET ?? %
OK:
  CODE 14,0;
        DECL$S
DERR:   MOV      $DSW,*RSXDSW/RRERRX(R0)
```

```
        BCC     *EXIT
  *RTL;
   RRGEL(14);   % DIRECTIVE ERROR       %
EXIT:
ENDPROC;

PROC FORCEDRELEASE(INT FA);

% SAME AS RELEASE, BUT BUT JUST RELEASES WITHOUT CHECKING WHETHER       %
% THE CALLER WAS SECURED. INTENDED FOR USE BY THE SYSTEM MANAGER TO     %
% 'UNHANG' A FACILITY WHICH HAS BEEN SECURED AND NOT SUBSEQUENTLY BEEN  %
% RELEASED.                                                             %

  REF R5ONAME TASK;
  RSXDSW:=FA;
  IF FA < 33 OR FA > 56 THEN
    RRGEL(14);
  END;
  TASK:=GLFACS(FA-32);

  %HERE WE LEAVE OUT THE CHECK OF USER IDENTITY %

  TASK.R5ON1 := TASK.R5ON2 := 0;

  CODE 20,0;
        .MCALL  SETF$S,DECL$S
        SETF$S  *FA(R5)
        BCS     DERR
        CMP     #IS.SET,$DSW
        BNE     *OK2
  *RTL;
  ERP(215);     % SECURED,BUT FLAG WAS SET ?? %
OK2:
  CODE 14,0;
        DECL$S
        MOV     $DSW,*RSXDSW/RRERRX(R0)
        BCC     *EXIT2
  *RTL;
   RRGEL(14);   % DIRECTIVE ERROR       %
EXIT2:
ENDPROC;

MODE IOSTAT (BYTE IOSTLOW,IOSTHIGH,INT IOSTVAL);

EXT PROC(INT,INT,INT,INT,REF IOSTAT,PROC(),REF ARRAY INT)RSXQIW;

DATA PRIVLOCAL;
  ARRAY(2)BYTE BUF;
  IOSTAT'STATUS;
  ARRAY(6)INT DP;
ENDDATA;


ENT PROC PRIVILEGED()INT;
  INT RET;
  REF BYTE RB:=BUF(1);
  INT BUFADR;
  INT FC;
  CODE 12,0;
        MOV     *RB(5),*BUFADR(5)       ;ADDR OF BUFFER
        MOV     #SF.GMC,*FC(5)  ;FN CODE GET MULT CHARACTERISTIX
  *RTL;
```

```
  DP(1):=BUFADR;
  DP(2):=2;
  DP(3):=0;
  DP(4):=0;
  DP(5):=0;
  DP(6):=0;

    BUF(1):=BYTE(OCT 51);        % CODE FOR FINDING PRIV OR NOPRIV        %
    BUF(2):=0;
    RSXQIW(FC,1,1,0,STATUS,RRNUL,DP);
    IF RSXDSW<0 THEN RRGEL(14);END;
    IF BUF(2)=1 THEN
      RET:=YES;
    ELSE
      RET :=NO;
    END;
  RETURN(RET);
ENDPROC;
```

N-8-1

```
LET MEDCOMEF = 33;

EXT PROC(INT) SET;

ENT PROC RRJOB();

% THIS IS TO BE CALLED AT SYSTEM STARTUP TO SET THE MEDCOM DATA-BASE    %
% SECURE/RELEASE EVENT FLAG, BECAUSE RSX BRINGS EVENT FLAGS UP AS CLEAR,%
% WHICH CORRESPONDS TO FACILITY ALREADY SECURED. IF THIS MEANS NOTHING  %
% TO YOU , TRY TYPING     >HELP RTL 214.                                %

SET(MEDCOMEF);

ENDPROC;
```

```
SVC DATA RRSED;
  BYTE TERMCH,IOFLAG;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT) ERP;
ENDDATA;

% MEDIA DATA BASE : %

EXT DATA INAREA;
  ARRAY(TOTALAI) MEDCARD ANINP;
  ARRAY(TOTALDIGICARD) MEDCARD DIGINP;
  ARRAY(2) MEDCARD DIGCHAN;
  MEDCARD MEDSTAT;
ENDDATA;

EXT DATA OUTAREA;
  ARRAY (TOTALAO) MEDCARD ANOUTP;
  ARRAY (TOTALDIGOCARD) MEDCARD DIGOUT;
  ARRAY (TOTALAO) AOREC AODESC;
  ARRAY (TOTALDIGOCARD,16) INT DIGUICS;
ENDDATA;

DATA LOCAL;
  INT MANAGER;
  INT QUIT := NO;
  ARRAY(9)BYTE UBUF;    % USED IN UICWRT        %
  PROC () DUMMY := RRNUL;
  TASKBUF TBUF := (0,0,0,0,0,0,0,0,0,0,0,DUMMY,0,0,0,0);
ENDDATA;

ENT PROC RRJOB();
  INT CHOICE;
  TTIO();
  TASKINFO(TBUF);       % GET INFO ABOUT USER   %
  IF TBUF.TASK17=OCT 140001 THEN        % [300,1] IS THE DATABASE MANAGER
%
    MANAGER:=YES;
  ELSE
    MANAGER:=NO;
  END;
  QUIT := NO;
  WHILE QUIT=NO DO
    HOME(); CLEOS();
    TWRT("#LF#MAIN MENU#LF,LF#");
    TWRT("1. ATTACH   #LF#");
    TWRT("2. DETACH#LF#");
    TWRT("3. QUIT#LF,LF,ENQ#");
    CHOICE:=IREAD();
    IF CHOICE=1 THEN
      ATTACH();
    ELSEIF CHOICE=2 THEN
      DETACH();
    ELSEIF CHOICE=3 THEN
      QUIT := YES;
    END;
  REP;
  HOME(); CLEOS(); OUT(ETX);
ENDPROC;
```

```
    RELMEDCOM();
   END;
  END;
  RETURN(RET);
ENDPROC;

ENT PROC WRITE ( INT MCODE,REF MEDCARD OUTDAT) INT;
                                % PROC TO WRITE DATA IN DATA BASE TO    %
                                % MEDIA.ONLY 1 CARD WRITTEN.            %
                                % ERROR STATUS IS RETURNED.             %

  INT RET:=0,STOP:=0,ERRCNT:=0;

  WHILE STOP = NO DO
   RET := SENDMESS(MCODE,OUTDAT.ADDR,OUTDAT.MEDDAT,1);
                                % SEND MESSAGE TO MEDIA - GET REPLY     %

   IF RET = NOERROR THEN
     RET:=SENDMESS(GOMES,OUTDAT.ADDR,0,0);% SEND 2ND MESSAGE - GET REPLY
%

     IF RET = NOERROR THEN              % NO ERRORS                     %
       SECMEDCOM();
       OUTDAT.SCANTIME:=TIMER(0.0);
       RELMEDCOM();
     END;
   END;
   IF RET = NOERROR THEN
     STOP := YES;                   % SUCESSFUL WRITE                 %
   ELSEIF RET = MEDIAERR THEN
     STOP := YES;                   % MEDIA ERROR                     %
   ELSE
     ERRCNT := ERRCNT + 1;
     IF ERRCNT >= 2 THEN            % 2 UNSUCCESSFUL RETRIES          %
       ERP (ERPBASE+RET);
       STOP:=YES;
     ELSE
       STOP := NO;                  % ERROR BUT TRY AGAIN             %
     END;
   END;
  REP;

  RETURN(RET);
ENDPROC;

ENT PROC GETMED (REF MEDCARD MSTATUS) INT;
       % PROC TO GET MEDIA STATUS WORD. IF THE RETURNED VALUE IS NOT   %
       % ZERO, THEN THE STATUS WORD COULD NOT BE ACCESSED.             %

   INT CHCK2;
   CHCK2:= SINGLIN(RSNGSTA,MSTATUS);% GET MEDIA STATUS WORD FROM MEDIA %
   RETURN(CHCK2);
ENDPROC;

ENT PROC GETDCW(REF ARRAY MEDCARD DIGCN) INT;
  % PROC TO GET THE DIGITAL CHANGE WORDS. IF THE RETURNED VALUE IS NOT  %
  % ZERO, THEN THE WORDS COULD NOT BE ACCESSED.                         %

  INT RET := 0;
  RET := BLOCKIN(RDIGCHG,1,2,DIGCN);
  RETURN(RET);
ENDPROC;
```

```
%   LOCAL PROCEDURES.                                                    %
%   =================                                                    %

PROC CHECINPUT (INT MCODE,NEXP) INT;
                                % PROC TO CHECK AN INPUT MESSAGE (IN     %
                                % INMSG) RECEIVED FROM MEDIA.THE RESULT %
                                % GIVES THE ERROR STATUS OF THE TRANSACTION.%
                                % NEXP IS EXPECTED LENGTH OF REPLY.      %

  INT NREC:=0;  % NO OF CHARS IN REPLY FROM MEDIA                        %
  INT RECSTATUS:=0;     % STATUS GOT FROM BYTE 2 OF RECEIVED MESSAGE     %
  INT TEMP:= NOERROR;
  BYTE T1:=0;

  IF MCODE=WSNGMED OR MCODE=WSNGANA THEN        % WRITE - ONLY NEED TO   %
    FOR I:=1 TO 6 DO                            % CHECK THAT REPLY IS    %
      IF INMSG(I)#OUTMSG(I) THEN                % SAME AS SENT MESSAGE. %
        TEMP := WRTERR;
      END;
    REP;
  ELSE                  % ALL CODES OTHER THAN WRITE     %
    NREC := GETNREC(NEXP);      % GET NO OF CHARS IN REPLY FROM MEDIA    %

    FOR I:=1 TO NREC-1 DO       % CALCULATE BCC CHARACTER OF RECEIVED BYTES %
      T1:=T1 NEV INMSG(I);      % BCC BY EXCLUSIVE OR                    %
    REP;
    T1:=(T1 LAND HEX 3F) LOR HEX 40;    % BITS 0-5 PLUS BIT 6=1          %
    IF T1 # (INMSG(NREC) LAND HEX 7F) THEN
      TEMP := BCCRECERR;        % BCC ERROR IN RECEIVED PACKET           %
    ELSE
      IF INMSG(1) LAND HEX 40 =0 OR INMSG(NREC) LAND HEX 40 = 0 THEN
        TEMP := ALIGNERR;       % ALIGNMENT ERROR IN RECEIEVD PACKET     %
      ELSE
        IF INMSG(1) LAND HEX F # MCODE THEN
          TEMP := WRONGCODE;    % SENT AND RECEIVED CODES DIFFER         %
        ELSE
          IF NREC=3 THEN        % ERROR REPLIES FROM MEDIA HAVE LENGTH 3 %
            RECSTATUS := INMSG(2) LAND HEX F;
            IF RECSTATUS  = 0 THEN      % NO STATUS ERROR                %
              IF NREC = NEXP THEN
                TEMP := NOERROR;        % NO ERROR                       %
              ELSE
                TEMP := LENERR; % ANSWER IS WRONG LENGTH          %
              END;
            ELSEIF RECSTATUS = 1 THEN
              TEMP:=PARERR;     % PARITY ERROR IN TRANSMITTED MESSAGE    %
            ELSEIF RECSTATUS = 2 THEN
              TEMP:=BLOCKERR;   % BCC ERROR IN TRANSMITTED MESSAGE %
            ELSEIF RECSTATUS = 8 THEN
              TEMP:=BADCODE;    % INVALID CODE SENT                      %
            ELSEIF RECSTATUS = 4 THEN
              TEMP:=MEDIAERR;   % MEDIA ERROR                            %
            ELSEIF RECSTATUS = HEX F THEN
              TEMP:=TIMEERR;    % TIMEOUT ERROR ON MEDIA ACCESS %
            ELSE
              TEMP:=ERRERR;     % UNKNOWN ERROR CODE                     %
            END;
          ELSE          % LENGTH OF INMSG IS NOT 3       %
            IF NREC=NEXP THEN
              TEMP := NOERROR;  % NO ERROR      %
            ELSE
              TEMP := LENERR;   % REPLY IS WRONG LENGTH %
```

```
          END;
        END;
      END;
    END;
  END;
  RETURN(TEMP);
ENDPROC;

PROC GETNREC(INT NEXP)INT;
  % PROC TO GET THE LENGTH OF THE ACTUAL RECEIVED MESSAGE FROM MEDIA    %
  % IT DOES THIS BY RETURNING THE NUMBER OF THE FIRST BYTE AFTER BYTE 1 %
  % WHICH HAS BIT 6 SET. IF NO SUCH BYTE IS FOUND WITHIN THE EXPECTED   %
  % LENGTH NEXP OF THE MESSAGE, THEN NEXP IS RETURNED.                  %

  INT  I,STOP;

  STOP := NO;
  I := 1;
  WHILE STOP = NO DO
    I := I + 1;
    IF INMSG(I) LAND BIN 01000000 = 0 THEN      % BIT 6 NOT SET         %
      IF I < NEXP THEN
        STOP := NO;                             % CONTINUE SEARCHING    %
      ELSE
        STOP := YES;            % CHAR WITH BIT 6 SET NOT FOUND WITHIN  %
                                % NEXP CHARS.                           %
      END;
    ELSE                        % BIT 6 IS SET.                         %
      STOP := YES;              % FOUND CHAR WITH BIT 6 SET.            %
    END;
  REP;
  RETURN(I);
ENDPROC;

PROC CHECLST (REF ARRAY MEDCARD MBLOCK,INT FIRST,LAST) INT;
                                % PROC TO CHECK THAT MEDIA DATA REQUESTED %
                                % FOR BLOCK INPUT IS CORRECTLY AND      %
                                % CONTIGUOUSLY ADDDRESSED               %

  INT RET := NOERROR;

  IF FIRST > 0 AND LAST-FIRST > 0 THEN
    FOR I:=FIRST+1 TO LAST DO
      IF MBLOCK(I-1).STAT LAND 2 = MBLOCK(I).STAT LAND 2 THEN
        % ADDRESSES ALL LIST OR ALL MEDIA                              %
        IF MBLOCK(I).ADDR # MBLOCK(I-1).ADDR + 1 THEN
          RET:=ADDRERR; % ADDRESSES NOT CONTIGOUSLY ORDERED           %
        END;
      ELSE                % ADDRESSES ARE MIXED LIST AND MEDIA        %
        RET:=MIXEDADDRS;
      END;
    REP;
  ELSE
    RET:=BADBLKLIMS;
  END;
  RETURN(RET);                  % SUCCESSFUL RETURN                     %
ENDPROC;

PROC DECODE ( INT I) INT;       % PROC TO  DECODE DATA IN INPUT BUFFER  %
                                % AND TO ASSEMBLE IT INTO AN INTEGER    %
```

```
    INT T1:=T2:=0;

    T1:=INMSG(3*I) LAND HEX 3F; % LEAST SIG 6 BITS IN BYTE 3*I          %
    T2:=INMSG(3*I+1) LAND HEX 1F;       % NEXT 5 BITS FROM NEXT BYTE    %
    T2:=T2 SLL 6;                % SHIFT 5 BITS TO CORRECT POSITION     %
    T1:=T1 LOR T2;               % LOGICAL OR WITH BITS ALREADY IN T1   %
    T2:=INMSG(3*I+2) LAND HEX 1F;       % GET LAST 5 BITS FROM NEXT BYTE %
    T2:=T2 SLL 11;               % SHIFT TO CORRECT POSITION            %
    T1:=T1 LOR T2;               % ADD THEM TO BITS ALREADY ASSEMBLED   %
    RETURN(T1);
ENDPROC;

PROC ENCODE (INT INP);
                % THIS PROC ENCODES INP INTO BYTES 4 TO 6 OF OUTMSG     %
  OUTMSG(4):=BYTE(INP LAND BIN 00111111);
  OUTMSG(5):=BYTE((INP SRL 6) LAND BIN 00011111);
  OUTMSG(6):=BYTE((INP SRL 11) LAND BIN 00011111);
ENDPROC;

PROC READ(INT MCODE,ADDRESS,NUMRD) INT;
                                % PROC TO DO DATA LINK HANDLING FOR A   %
                                % READ MEDIA TRANSACTION                %

    INT TEMP,ERRCNT:=0,STOP:=NO;

    WHILE STOP = NO DO
        TEMP := SENDMESS(MCODE,ADDRESS,NUMRD,NUMRD); % SEND OUT MESSAGE AND %
                                % GET REPLY, AND CHECK IT.
        %

        IF TEMP # NOERROR THEN  % AN ERROR OF SOME SORT OCCURRED        %
            IF TEMP # MEDIAERR THEN     % NOT MEDIA ERROR              %
                ERRCNT := ERRCNT + 1;
                IF ERRCNT >= 2 THEN     % 2 RETRIES HAVE BEEN ATTEMPTED %
                    ERP(ERPBASE+TEMP);  % ABORT WITH DATA LINK ERROR NUMBER %
                    STOP := YES;                % NO MORE RETRIES       %
                ELSE
                    STOP := NO;         % RETRY                         %
                END;
            ELSE
                STOP := YES;            % MEDIA ERROR SO NO RETRIES     %
            END;
        ELSE
            STOP :=YES;                 % NO ERRORS - SUCCESSFUL RETURN %
        END;
    REP;
    RETURN(TEMP);
ENDPROC;

PROC SENDMESS(INT MCODE,MADR,MDATA,NUMRD)INT;

% SETS UP MESSAGE TO BE SENT TO MEDIA, SENDS IT, GETS A REPLY FROM MEDIA %
% AND CHECKS IT FOR ERRORS. THE VALUE RETURNED IS THE ERROR STATUS      %

  INT NIN:=0;           % NO OF CHARS EXPECTED IN REPLY          %
  INT NOUT:=0;          % NO OF CHARS IN OUTPUT MESSAGE          %
  INT TEMP:=0;

  OUTMSG(1) := BYTE(MCODE) LOR BIN 01000000;

  IF MCODE = RSNGMED OR MCODE = RSNGANA THEN    %SINGLE READ    %
    ADDRESS(MADR);
    NOUT := 4;
```

```
    NIN := 6;
  ELSEIF MCODE = RBLKMED OR MCODE = RBLKANA THEN
    ADDRESS(MADR);
    NOUT:=5;
    NIN:=3*(NUMRD + 1);
    OUTMSG(4):=BYTE(NUMRD LAND HEX 003F);
  ELSEIF MCODE = RDIGCHG THEN                    % READ DCW'S          %
    OUTMSG(2):=0;
    OUTMSG(3):=0;
    NOUT:=4;
    NIN:=9;
  ELSEIF MCODE = RSNGSTA THEN                    % READ MEDIA STATUS   %
    OUTMSG(2):=0;
    OUTMSG(3):=0;
    NOUT:=4;
    NIN:=6;
  ELSEIF MCODE = WSNGMED OR MCODE = WSNGANA THEN      % WRITE %
    ADDRESS(MADR);
    NOUT:=7;
    NIN:=7;
    ENCODE(MDATA);
  ELSEIF MCODE = GOMES THEN                      % 2ND STAGE OF WRITE  %
    ADDRESS(SAVEADDR);
    NOUT:=4;
    NIN:=3;
  ELSE                                           % CODE NOT USED       %
    OUTMSG(2):=0;
    NOUT:=3;
    NIN:=3;
  END;

  SAVEADDR := MADR;

  PUTBCC(NOUT);

  FOR I:=1 TO NIN DO            % SET INMSG TO ZERO'S   %
    INMSG(I) := 0;
  REP;
  MESSANS(OUTMSG,NOUT,INMSG,NIN);       % SEND MESSAGE, GET REPLY      %

  TEMP := CHECINPUT(MCODE,NIN); % CHECK REPLY FOR ERRORS        %
  RETURN(TEMP);
ENDPROC;

PROC ADDRESS(INT ADR);
                % ENCODES ADR INTO BYTES 2 AND 3 OF OUTMSG              %
  OUTMSG(2):= BYTE(ADR LAND HEX 000F);
  OUTMSG(3):= BYTE((ADR SRL 4) LAND HEX 003F);
ENDPROC;

PROC PUTBCC(INT MSGLEN);
  INT T:=0;
  BYTE B;
  FOR I:=1 TO MSGLEN - 1 DO
    T:= T NEV OUTMSG(I);
  REP;
  B:=BYTE(T LAND OCT 077);
  B:=B LOR BIN 01000000;
  OUTMSG(MSGLEN):=B;
ENDPROC;
```

```
TITLE GSECREL
;
%  SECURE AND RELEASE PROCS USING GLOBAL EVENT FLAGS.    %

LET YES = 1;
LET NO = 0;
LET MEDCOMEF = 33;       % EVENT FLAG FOR MEDCOM SECURE/RELEASE  %

MODE R5ONAME(INT R5ON1,R5ON2);
MODE TASKBUF(INT TASKI1,TASKI2,
                 TASKP1,TASKP2,
                 TASKR1,TASKR2,
                 TASKRUN,
                 TASKUIC,         % CURRENT UIC %
                 TASKNLUN,
                 TASKMCTY,
                 TASKSTDF,
                 REF PROC() TASTSST,
                 INT TASTSST2,
                 TASK15,TASK16,
                 TASK17);         % PROTECTION UIC        %

EXT PROC (INT) SET;      % SET EVENT FLAG         %
EXT PROC () RRNUL;
EXT PROC (INT) RRGEL;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT ) OWRT;
EXT PROC () TTIO;
EXT PROC(REF R5ONAME)R5OREAD,R5OWRT;

SVC DATA RRERRX;
  INT LINENO;
  BYTE UEFLAG,ERRLUN;
  INT RSXDSW;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT) ERP;
ENDDATA;

SVC DATA RRTASK;
  R5ONAME MYTASK;
ENDDATA;

EXT DATA RRGLFACS;
  ARRAY(24) R5ONAME GLFACS;
ENDDATA;

DATA LOCAL;
  PROC() DUMMY:=RRNUL;
  TASKBUF TBUF:=(0,0, 0,0, 0,0, 0,0,0,0,0,DUMMY,0,0,0,0);
ENDDATA;

ENT PROC SECMEDCOM();

% PROC TO SECURE THE MEDCOM DATA BASE. IT CALLS SECURE USING EVENT      %
% FLAG MEDCOMEF.                                                        %

  SECURE(MEDCOMEF);
ENDPROC;
```

```
ENDPROC;

PROC GETBLOCK();
  % READS A 512 BYTE BLOCK OF DATA FROM THE FILE OPENED BY CALLING      %
  % OPENBLKIN, AND PUTS IT INTO IOBUF. CALLS ERP IF END-OF-FILE IS DETECTED %
  INT S;
  S:=GETBLK(BUFADR,1,0,512);
  IF S<0 THEN RRGEL(605);END;
  IF S=1 THEN ERP(604);END;
ENDPROC;
```

```
;
;    SMTLOAD
;    =======
;
;    BOOTSTRAP LOADER FOR THE LSI-11 TO LOAD A PROGRAM DOWN
;    THE CONSOLE SERIAL LINE.   SMTLOAD MUST BE RELOCATED
;    TO 157000=START ADDRESS AT TASKBUILD TIME, BY BUILDING AS FOLLOWS:
;
;      TKB>SMTLOAD/-MM/-HD,SMTLOAD/-SP=SMTLOAD
;      TKB>/
;      TKB>STACK=0
;      TKB>PAR=EXEPAR:157000:1000
;      TKB>//
;
;    SMTLOAD.ABS (THE FORM IN WHICH SMTLOAD IS USED BY ODT.TSK) IS MADE
;    AS FOLLOWS:
;      RUN $DMP
;      DMP>SMTLOAD.ABS=SMTLOAD.TSK/BL:3
;    THEN SMTLOAD.ABS IS EDITED (USING EDT) TO REMOVE ALL <LF> CHARACTERS,
;    DELETE THE FIRST 2 LINES AND DELETE ALL LINES CONTAINING ONLY ZERO
;    DATA.
;
;
        .PSECT
;
;        DEFINITIONS
;        ***********
;
RCSR    = 177560          ;RECEIVER CONTROL/STATUS REGISTER
RBUF    = RCSR+2          ;READ BUFFER ADDRESS
;
XCSR    = 177564          ;XMIT CONTROL/STATUS REGISTER
XBUF    = XCSR+2          ;XMIT BUFFER ADDRESS
;
STACK   = 157000          ;STACK POINTER (HIGHEST ADDRESS)
;
BLOCKSIZE = 1000          ;SIZE OF BLOCK OF INPUT DATA
;
; THIS PROGRAM MUST BE EXECUTED WITH THE PSW SET TO 340 (PRIORITY 7).
; THIS IS DONE IN ODT BY LOADING THE PC WITH THE START ADDRESS
; (157000) AND THEN GIVING THE PROCEED (P) COMMAND. NOTE THAT THE
; COMMAND '1570006' WILL NOT WORK AS THE GO (G) COMMAND CLEARS THE
; PSW.

BEGIN:
        MOV     #STACK,SP       ;SET UP STACK POINTER
;
        JSR     PC,GETCHAR              ;GET THE FIRST TWO CHARS. THESE ARE
        MOVB    @#CHAR,@#LENGTH         ;THE NO OF BYTES TO FOLLOW (IE LENGTH
        JSR     PC,GETCHAR
        MOVB    @#CHAR,@#LENGTH+1
;
        CLR     R3      ; I
        CLR     R2
        CLRB    @#BCC
;
FOR:                                    ;FOR I:=0 TO LENGTH-1 DO
        JSR     PC,GETCHAR              ;  GET NEXT INPUT CHARACTER
        MOVB    @#CHAR,R1
        MOVB    R1,(R3)                 ;    STORE IT AT ADDRESS I;
;
        ADD     @#BCC,R1                ;  BCC:=BCC + CHAR;
```

```
        MOVB    R1,@#BCC
;
        INC     R2
;
        CMP     R2,#BLOCKSIZE           ;  IF BLOCKCOUNT >= BLOCKSIZE THEN
        BLO     ENDIF
        CLR     R2                      ;    BLOCKCOUNT:=0;
;
LOOP:   TSTB    @#XCSR
        BPL     LOOP
        MOVB    @#BCC,@#XBUF            ;    OUTPUT THE BCC;
;
        CLRB    @#BCC                   ;    BCC:=0;
ENDIF:                                  ;  END;
        INC     R3
        CMP     R3,@#LENGTH
        BLO     FOR                     ;REP;
;
        HALT
;
;
GETCHAR:                                ;PROCEDURE TO INPUT THE NEXT BYTE
        TSTB    @#RCSR                  ;  LOOP TILL CHAR HAS ARRIVED
        BPL     GETCHAR
        MOVB    @#RBUF,@#CHAR
        RTS     PC
;
                        ;VAR
;
                        ; I: ADDRESS COUNTER -- USE R3
                        ; BLOCKCOUNT: COUNT OF CHARS IN PRESENT BLOCK -- USE
LENGTH: .WORD   0       ; NO OF CHARS TO BE READ IN
BCC:    .BYTE   0       ; BLOCK CHECK CHAR
CHAR:   .BYTE   0       ; BYTE CHAR : CHARACTER RECEIVED

        .END
```

```
OPTION (1) CM;
TITLE
MEDCOM OCT-1981;

%                                                                    %
%   DESCRIPTION OF MODULE.                                           %
%   ======================                                           %
%    THIS IS THE COMMON DATA BASE FOR ACCESSING MEDIA.               %
%    THE DATA IN MEDIA IS COPIED INTO THIS DATA BASE BY THE          %
%    MEDIA UPDATE TASK. FOR EACH INPUT AND OUTPUT, A RECORD OF       %
%    THE TIME OF LAST UPDATE IS MAINTAINED.                          %
%                                                                    %

%   LET DEFINITIONS                                                  %
%   ===============                                                  %

LET NMULT = 16;                          % NUMBER OF MULTIPLEXED AN. INPUTS %
LET NSIMAI = 4;                          % NUMBER OF SIMULATED AN. INPUTS   %
LET TOTALAI = 20;

LET NDIGICARD = 2;                       % NO OF MEDIA DGITAL INPUT CARDS   %
LET NSIMDIGICARD = 2;                    % NO OF SMULATION DIG. INPUT CARDS %
LET TOTALDIGICARD = 4;

LET NAO = 4;                             % NO OF MEDIA ANALOG OUTPUTS       %
LET NSIMAO = 4;                          % NO OF SIMULATION ANALOG OUTPUTS  %
LET TOTALAO = 8;

LET NDIGOCARD = 1;                       % NO OF MEDIA DIG. OUTPUT CARDS    %
LET NSIMDIGOCARD = 1;                    % NO OF MEDIA DIG. INPUT CARDS     %
LET TOTALDIGOCARD = 2;

% THESE MUST BE CHANGED IF MEDIA CARDS ARE ADDED                           %

%   MODE DEFINITIONS.                                                %
%   =================                                                %

MODE MEDCARD (                   % RECORD FOR STORING DESCRIPTION OF     %
                                 % A SINGLE MEDIA CARD OR ANALOGUE INPUT %
              INT STAT,          % STATUS WORD                           %
                     % BIT 0 : 1 = IN SCAN.  0 = OUT OF USE             %
                     % BIT 1 : 1 = 'ADDR' IS MEDIA ADDRESS              %
                     %          0 = 'ADDR' IS LIST ADDRESS              %
                     % BIT 2 : 1 = OUTPUT .  0 = INPUT.                 %
                     % BIT 3 : 1 = ANALOGUE (I.E. FRAC-INT CONV.REQUIRED %
                     %          0 = DIGITAL                             %
                     % BIT 4 : 1 = ANALOG IS BOTH +VE AND -VE           %
                     %          0 = ANALOG IS +VE ONLY                  %
                     % BIT 5 : 1 = CARD REPRESENTS PART OF MEDIA        %
                     %          0 = CARD IS USED FOR SIMULATION         %
                 MEDDAT,         % DATA AS OBTAINED FROM MEDIA           %
                 ADDR,           % MEDIA OR LIST ADDRESS OF CARD OR INPUT %
              REAL SCANTIME      % TIME OF LAST TRANSACTION BETWEEN DATA %
                                 % BASE AND MEDIA FOR THIS CARD OR INPUT %
                     );

MODE AOREC (                     % RECORD FOR DESCRIBING AN ANALOGUE OUTPUT  %
            INT UIC,             % UIC OF USER                               %
            USERINT,             % INTEGER FOR THE USER TO USE               %
                                 % FOR INTER-TASK COMMUNICATION.            %
            ANIN,                % AI WHICH IS BEING                         %
                                 % CONTROLLED BY THIS AO                    %
```

```
            REAL SETP   % SETPOINT OF THIS AI                          %
                    );

%   ENT DATA BRICK DEFINITIONS.                                        %
%   ===========================                                        %

ENT DATA INAREA;        % DATA BRICK FOR HOLDING COPIES OF MEDIA INPUTS %
    ARRAY (TOTALAI) MEDCARD ANINP:=% ANALOGUE INPUTS %
                                ((HEX 0009,0,1,0.0),
                                 (HEX 0009,0,2,0.0),
                                 (HEX 0009,0,3,0.0),
                                 (HEX 0009,0,4,0.0),
                                 (HEX 0009,0,5,0.0),
                                 (HEX 0009,0,6,0.0),
                                 (HEX 0009,0,7,0.0),
                                 (HEX 0009,0,8,0.0),
                                 (HEX 0009,0,9,0.0),
                                 (HEX 0009,0,10,0.0),
                                 (HEX 0009,0,11,0.0),
                                 (HEX 0009,0,12,0.0),
                                 (HEX 0009,0,13,0.0),
                                 (HEX 0009,0,14,0.0),
                                 (HEX 0009,0,15,0.0),
                                 (HEX 0009,0,16,0.0),

                                 (HEX 0029,0,0,0.0), % THESE ARE SIMULATED %
                                 (HEX 0029,0,0,0.0),
                                 (HEX 0029,0,0,0.0),
                                 (HEX 0029,0,0,0.0));

    ARRAY (TOTALDIGICARD) MEDCARD DIGINP:=      % DIGITAL INPUTS        %
                                ((HEX 0003,0,0,0.0),
                                 (HEX 0003,0,1,0.0),

                                 (HEX 0023,0,0,0.0),    % SIMULATED     %
                                 (HEX 0023,0,0,0.0));

    ARRAY (2) MEDCARD DIGCHAN:=          % DIGITAL INPUT CHANGE WORD     %
                               ((3,0,0,0.0),(3,0,0,0.0));
    MEDCARD MEDSTAT;           % STATUS WORD FOR MEDIA                   %
ENDDATA;

ENT DATA OUTAREA;       % DATA BRICK HOLDING DATA TO BE SENT TO MEDIA   %
    ARRAY (TOTALAO) MEDCARD ANOUTP:=    % ANALOGUE OUTPUTS               %
                                ((HEX 000F,HEX 0001,OCT 14,0.0),
                                 (HEX 000F,HEX 0002,OCT 14,0.0),
                                 (HEX 000F,HEX 0004,OCT 14,0.0),
                                 (HEX 000F,HEX 0008,OCT 14,0.0),

                                 (HEX 002F,HEX 0001,0,0.0), % SIMULATED %
                                 (HEX 002F,HEX 0002,0,0.0),
                                 (HEX 002F,HEX 0004,0,0.0),
                                 (HEX 002F,HEX 0008,0,0.0));


    ARRAY (TOTALDIGOCARD) MEDCARD DIGOUT:=      % DIGITAL OUTPUTS        %
                                ((HEX 0007,0,4,0.0),

                               (HEX 0027,0,0,0.0));     % SIMULATED %

    ARRAY (TOTALAO) AOREC AODESC:= ((0,0,0,0.0)(TOTALAO));
                                % DESCRIPTIONS OF AO'S                    %
```

```
    ARRAY (TOTALDIGOCARD,16) INT DIGUICS;          %DESCRIPTIONS OF DO'S
%

ENDDATA;
```

(N-4-1)

```
TITLE MEDUPDAT
      MEDIA UPDATE TASK;

LET     LF              = OCT 12;
LET     BEL             = 7;
LET     ETX             = 3;    % CONTROL-C END-OF-TEXT %

LET     DEL             = 10;   % DELAY BETWEEN UPDATES                    %
LET     MEDIAERR        = 7;    % MEDIA ERROR                             %
LET     NOERROR         = 0;

LET     RSNGMED         = 1;    % SINGLE READ FROM MEDIA                  %
LET     RBLKMED         = 3;    % BLOCK READ FROM MEDIA                   %
LET     RBLKANA         = 4;    % BLOCK LIST READ                         %
LET     WSNGMED         = 12;   % WRITE TO MEDIA                          %

MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);
MODE AOREC  (INT UIC, USERINT, ANIN,REAL SETPOINT);

EXT PROC(INT) DELAY;
EXT PROC() SECMEDCOM,RELMEDCOM,FREEMEDCOM,MCOMINIT;  % IN SECREL.RTL %
EXT PROC (INT,REF MEDCARD) INT SINGLIN;
EXT PROC (INT,INT,INT,REF ARRAY MEDCARD) INT BLOCKIN;
EXT PROC (REF ARRAY MEDCARD) INT GETDCW;
EXT PROC (INT,REF MEDCARD) INT WRITE;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (BYTE)TTOUT;
EXT PROC() TTIO,CLEANUP;
EXT PROC (INT) IWRT,OWRT;

SVC DATA RRSIO;
  PROC() BYTE IN;
  PROC(BYTE) OUT;
ENDDATA;

SVC DATA RRSED;
  BYTE TERMCH,IOFLAG;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT) ERP;
ENDDATA;

SVC DATA RRERRX;
  INT LINENO;
  BYTE UBFLAG,ERRLUN;
  INT RSXDSW;           % HOLDS DSW RESULT OF EXECUTIVE CALLS   %
ENDDATA;

%                                                                %
% MEDCOM DATA BASE                                               %
%                                                                %

LET NMULT = 16;                         % NUMBER OF MULTIPLEXED AN. INPUTS %
LET NSIMAI = 4;                         % NUMBER OF SIMULATED AN. INPUTS   %
LET TOTALAI = 20;

LET NDIGICARD = 2;                      % NO OF MEDIA DGITAL INPUT CARDS %
LET NSIMDIGICARD = 2;                   % NO OF SMULATION DIG. INPUT CARDS %
LET TOTALDIGICARD = 4;
```

```
LET NAO = 4;                                % NO OF MEDIA ANALOG OUTPUTS    %
LET NSIMAO = 4;                             % NO OF SIMULATION ANALOG OUTPUTS %
LET TOTALAO = 8;

LET NDIGOCARD = 1;                          % NO OF MEDIA DIG. OUTPUT CARDS %
LET NSIMDIGOCARD = 1;                       % NO OF MEDIA DIG. INPUT CARDS  %
LET TOTALDIGOCARD = 2;

% THESE MUST BE CHANGED IF MEDIA CARDS ARE ADDED                           %


EXT DATA INAREA;        % DATA BRICK FOR HOLDING COPIES OF MEDIA INPUTS %
    ARRAY (TOTALAI) MEDCARD ANINP;
    ARRAY (TOTALDIGICARD) MEDCARD DIGINP;
    ARRAY (2) MEDCARD DIGCHAN;
    MEDCARD MEDSTAT;            % STATUS WORD FOR MEDIA                  %
ENDDATA;

EXT DATA OUTAREA;       % DATA BRICK HOLDING DATA TO BE SENT TO MEDIA   %
    ARRAY (TOTALAO) MEDCARD ANOUTP;
    ARRAY (TOTALDIGOCARD) MEDCARD DIGOUT;
    ARRAY (TOTALAO) AOREC AODESC;
    ARRAY (TOTALDIGOCARD,16) INT DIGUICS;
ENDDATA;

DATA LOCAL;
  INT ERRCODE := NOERROR;
ENDDATA;

ENT PROC RRJOB();

  INT LOOPCNT:=1;
  ERL:=UNRECOV;

  TTIO();

  TWRT("#LF#MEDIA DATA BASE UPDATE TASK V2.1#LF,ETX#");

  % SET THE MEDIA SECURE/RELEASE EVENT FLAG (SEE COMMENTS IN SECREL.RTL) %
  MCOMINIT();

STARTUPDAT:
  WHILE 1=1 DO

    RDANINPS();         % READ ANALOG INPUTS AND PUT OUT OF SCAN IF     %
                        % MEDIA ERROR OCCURS.                          %

    FOR I:=1 TO NDIGICARD DO
      RDDIGINP(I);      % READ DIGITAL INPUTS, ETC...                   %
    REP;

%    RDDCWS();          %% READ DIG CHG WDS, DITTO...                   %

    FOR I:=1 TO NAO DO
      WRANOUT(I);               % WRITE TO ANALOG OUTPUTS, DITTO
%
    REP;

    FOR I:=1 TO NDIGOCARD DO
      WRDIGOUT(I);      % WRITE TO DIG O/P'S, DITTO                     %
    REP;
```

```
    DELAY(DEL);

    LOOPCNT := (LOOPCNT + 1) MOD 10;
    IF LOOPCNT=0 THEN
      CHECKSCAN();       % CHECK TO SEE IF ANY CARDS HAVE COME   %
    END;                 % BACK INTO SCAN.                       %

  REP;

UNRECOV:          % UNRECOVERABLE ERROR HANDLING.                 %
  FREEMEDCOM(); % FORCED RELEASE OF MEDCOM IRRESPECTIVE OF WHO  %
                % HAS SECURED IT.                                %
  CLEANUP();
  TTIO();
  GOTO STARTUPDAT;
ENDPROC;


PROC RDANINPS();

  % READ IN THE ANALOGUE INPUTS                                  %

  IF ANINP(1).STAT LAND 1 = 1 THEN       % IN SCAN       %
    ERRCODE := BLOCKIN(RBLKANA,1,NMULT:/2,ANINP);
    IF ERRCODE#MEDIAERR THEN
      ERRCODE:=BLOCKIN(RBLKANA,NMULT:/2 +1,NMULT,ANINP);
    END;
    IF ERRCODE=MEDIAERR THEN
      SECMEDCOM();
      FOR I:=1 TO NMULT DO
        ANINP(I).STAT := ANINP(I).STAT LAND HEX FFFE;   % PUT OUT OF SCAN %
      REP;
      RELMEDCOM();
      FOR I:=1 TO NMULT DO
        TWRT("#LF#OUT OF SCAN : ANINP "); IWRT(I);TWRT("#BEL,ETX#");
      REP;
    END;
  END;

ENDPROC;

PROC RDDIGINP(INT N);

  % READ IN THE DIGITAL INPUTS                                   %

  IF DIGINP(N).STAT LAND 1 = 1 THEN      % IN SCAN       %
    ERRCODE := SINGLIN(RSNGMED,DIGINP(N));
    IF ERRCODE=MEDIAERR THEN
      SECMEDCOM();
      DIGINP(N).STAT := DIGINP(N).STAT LAND HEX FFFE;
      RELMEDCOM();
      TWRT("#LF#OUT OF SCAN : DIGINP "); IWRT(N);TWRT("#BEL,ETX#");
    END;
  END;

ENDPROC;

PROC RDDCWS();

  % READ IN THE DIGITAL CHANGE WORDS                                %

  IF DIGCHAN(1).STAT LAND 1 = 1 AND DIGCHAN(2).STAT LAND 1 = 1 THEN
```

```
    ERRCODE := GETDCW(DIGCHAN);
    IF ERRCODE=MEDIAERR THEN
      SECMEDCOM();
      FOR I:=1 TO 2 DO
        DIGCHAN(I).STAT := DIGCHAN(I).STAT LAND HEX FFFE;
      REP;
      RELMEDCOM();
      TWRT("#LF#OUT OF SCAN : DCWS");TWRT("#BEL,ETX#");
    END;
  END;

ENDPROC;

PROC WRANOUT(INT N);

  % WRITE TO ANALOGUE OUTPUT N                                          %

  IF ANOUTP(N).STAT LAND 1 = 1 THEN
    ERRCODE := WRITE(WSNGMED,ANOUTP(N));
    IF ERRCODE=MEDIAERR THEN
      SECMEDCOM();
      ANOUTP(N).STAT := ANOUTP(N).STAT LAND HEX FFFE;
      RELMEDCOM();
      TWRT("#LF#OUT OF SCAN : ANOUT "); IWRT(N);TWRT("#BEL,ETX#");
    END;
  END;

ENDPROC;

PROC WRDIGOUT(INT N);

  % WRITE TO THE DIGITAL OUTPUTS                                        %

  IF DIGOUT(N).STAT LAND 1 = 1 THEN
    ERRCODE := WRITE(WSNGMED,DIGOUT(N));
    IF ERRCODE=MEDIAERR THEN
      SECMEDCOM();
      DIGOUT(N).STAT := DIGOUT(N).STAT LAND HEX FFFE;
      RELMEDCOM();
      TWRT("#LF#OUT OF SCAN : DIGOUT "); IWRT(N);TWRT("#BEL,ETX#");
    END;
  END;

ENDPROC;


PROC CHEGKSCAN();

  % CHECKS ALL CARDS THAT HAVE GONE OUT OF SCAN TO SEE IF THEY  %
  % CAN BE BROUGHT BACK INTO SCAN.                              %


  IF ANINP(1).STAT LAND 1 = 0 THEN      % OUT OF SCAN   %
    CHANINPS(); % TRY TO ACCESS THE ANALOG INPUTS       %
  END;

  FOR I:=1 TO NDIGICARD DO
    IF DIGINP(I).STAT LAND 1 = 0 THEN   % OUT OF SCAN   %
      CHDIGINP(I);
    END;
  REP;
```

```
%  IF DIGCHAN(1).STAT LAND 1 = 0 THEN   %% OUT OF SCAN  %
%    CHDCWS();%
%  END;%

  FOR I:=1 TO NAO DO
    IF ANOUTP(I).STAT LAND 1 = 0 THEN   % OUT OF SCAN   %
      CHANOUT(I);
    END;
  REP;

  FOR I:=1 TO NDIGOCARD DO
    IF DIGOUT(I).STAT LAND 1 = 0 THEN   % OUT OF SCAN   %
      CHDIGOUT(I);
    END;
  REP;

ENDPROC;

PROC CHANINPS();

% CHECK IF ANINPS BACK IN SCAN  %

  ERRCODE := BLOCKIN(RBLKANA,1,NMULT,ANINP);
  IF ERRCODE#MEDIAERR THEN       % BACK IN SCAN  %
    SECMEDCOM();
    FOR I:=1 TO NMULT DO
      ANINP(I).STAT := ANINP(I).STAT LOR 1;
    REP;
    RELMEDCOM();
    FOR I:=1 TO NMULT DO
      TWRT("#LF#BACK IN SCAN : ANINP "); IWRT(I); TWRT("#BEL,ETX#");
    REP;
  END;

ENDPROC;

PROC CHDIGINP(INT N);

  % CHECKS IF DIGINP N HAS COME BACK INTO SCAN  %

  ERRCODE := SINGLIN(RSNGMED,DIGINP(N));
  IF ERRCODE#MEDIAERR THEN
    SECMEDCOM();
    DIGINP(N).STAT:=DIGINP(N).STAT LOR 1;
    RELMEDCOM();
    TWRT("#LF#BACK IN SCAN : DIGINP "); IWRT(N); TWRT("#BEL,ETX#");
  END;

ENDPROC;

PROC CHDCWS();

% CHECK IF DCWS ARE BACK IN SCAN        %

  ERRCODE:=GETDCW(DIGCHAN);
  IF ERRCODE#MEDIAERR THEN
    SECMEDCOM();
    FOR I:=1 TO 2 DO
      DIGCHAN(I).STAT:=DIGCHAN(I).STAT LOR 1;
    REP;
    RELMEDCOM();
    TWRT("#LF#BACK IN SCAN : DCWS#BEL,ETX#");
```

N-4-5

```
  END;
ENDPROC;

PROC CHANOUT(INT N);

% CHECK IF ANOUT N IS BACK IN SCAN      %

  ERRCODE:=WRITE(WSNGMED,ANOUTP(N));
  IF ERRCODE # MEDIAERR THEN
    SECMEDCOM();
    ANOUTP(N).STAT:=ANOUTP(N).STAT LOR 1;
    RELMEDCOM();
    TWRT("#LF#BACK IN SCAN : ANOUT "); IWRT(N); TWRT("#BEL,ETX#");
  END;

ENDPROC;


PROC CHDIGOUT(INT N);

% CHECKS IF DIGOUT N HAS COME BACK IN SCAN      %

  ERRCODE:=WRITE(WSNGMED,DIGOUT(N));
  IF ERRCODE#MEDIAERR THEN
    SECMEDCOM();
    DIGOUT(N).STAT:=DIGOUT(N).STAT LOR 1;
    RELMEDCOM();
    TWRT("#LF#BACK IN SCAN : DIGOUT "); IWRT(N); TWRT("#BEL,ETX#");
  END;

ENDPROC;
```

```
OPTION (1) CM;

TITLE
LINKLB;

%                                                                       %
%    THIS MODULE CONTAINS PROCEDURES FOR PASSING AND RECEIVING DATA     %
%    LINK PROCEDURES IN RSX 11M. IT IS INTENDED TO BUILD IT INTO A      %
%    GENERAL LIBRARY, ALTHOUGH IT IS INITIALLY WRITTEN TO SERVE THE     %
%    MEDIA HANDLING PROCEDURES.                                         %
%              THE ENT PROC DEFINED HERE IS:                            %
%       MESSANS :       PROC TO WRITE BUFFER DOWN DATA LINK AND TO READ %
%                       IN REPLY BUFFER,BOTH WITH TIMEOUT               %
%                                                                       %

LET LUN = 3;                    % LOGICAL UNIT NUMBER OF MEDIA          %
LET RAL = OCT 001010;           % COMMAND CODE FOR READ ALL              %
LET RNE = OCT 20;               % COMMAND SUB-CODE FOR READ NO ECHO      %
LET WAL = OCT 000410;           % COMMAND CODE FOR WRITE ALL             %
LET KIL = OCT 000012;           % CANCEL I/O REQUEST COMMAND CODE        %
LET BUFLENERR = 13;             % BUFFER LENGTH ERROR                    %
LET TIMERRI = 14;               % TIMEOUT ERROR ON INPUT                 %
LET TIMERRO = 15;               % TIMEOUT ERROR ON OUTPUT                %
LET INERR = 16;                 % QIO STATUS ERROR ON INPUT              %
LET OUTERR = 17;                % QIO STATUS ERROR ON OUTPUT             %
LET NOERROR = 0;                % NO ERROR DETECTED                      %
LET LF = OCT 012;               % LINE FEED CHARACTER                    %
LET SP = OCT 040;               % SPACE CHARACTER                        %
LET BIT6 = BIN 01000000;                % BIT 6 MASK
%

%   MODE DEFINITIONS.                                                   %
%   =================                                                   %

MODE IOSTAT (BYTE IOSTLOW, IOSTHIGH, INT IOSTVAL);      % IO STATUS BLOCK %
MODE FLAGBUF (INT FLAGLOCAL0,FLAGLOCAL1,FLAGCOMMON2,FLAGCOMMON3);
                                % EVENT FLAG SETTINGS FOR RSX11M DIRS.  %

%   EXTERNAL PROCEDURE DEFINITIONS.                                     %
%   ===============================                                     %

% THE FOLLOWING ROUTINES ARE IN THE BASE PROGRAMS                       %
EXT PROC () RRNUL;              % NULL PROCEDURE                        %
EXT PROC ( INT ) RRGEL;         % UNRECOVERABLE ERROR HANDLER           %
%   THE FOLLOWING ARE ROUTINES FROM LB:(1,1)MTSLIB.OLB                  %
EXT PROC ( INT ) RELDEV,SECDEV; % DEVICE RELEASE,SECURE                 %
EXT PROC ( INT,INT,INT ) MARKTIME;
%   THE POLLOWING ARE STANDARD STREAM I/O PROCEDURES - REFER            %
%   TO THE STREAM I/O MANUAL                                            %

EXT PROC ( REF ARRAY BYTE ) TWRT;
EXT PROC ( INT ) IWRT,OWRT;

%   THE FOLLOWING ARE RSX 11M EXECUTIVE DIRECTIVES FROM                 %
%   LB:(1,1)RTLEXC.OLB                                                  %
EXT PROC ( INT,INT,INT,INT,REF IOSTAT,PROC(),REF ARRAY INT ) RSXQIW;
EXT PROC (INT,PROC()) RSXCMK;   % CANCEL MARK TIME REQUESTS             %

%   SVC DATA BRICK DEFINITIONS                                          %
%   ==========================                                          %

SVC DATA RRERR;
    LABEL ERL;
```

```
    INT ERN;
    PROC ( INT ) ERP;                                        (N-5-2)
ENDDATA;

%   LOCAL DATA BRICK DEFINITIONS.                                    %
%   =============================                                    %

DATA LOCAL;
    IOSTAT STATUS;              % IO STATUS BLOCK                    %
    ARRAY (6) INT DEVPARM;      % DEVICE SPECIFIC PARAMETERS FOR QIO  %
    INT ERRCODE:=NOERROR;       % ERROR CODE OF LINK TRANSACTION      %
ENDDATA;

%   ENT PROCEDURES.                                                  %
%   ===============                                                  %

ENT PROC MESSANS (REF ARRAY BYTE OUTBUF,INT OUTLEN,REF ARRAY BYTE INBUF,
                  INT INLEN)INT;

  % THIS PROC OUTPUTS 'OUTLEN' CHARS DOWN THE LINE AND WAITS FOR       %
  % A REPLY OF 'INLEN' CHARS.   IT FIRST WAITS FOR THE FIRST THREE     %
  % CHARS OF THE REPLY, SINCE ALL ERROR REPLIES FROM MEDIA ARE THREE   %
  % CHARS LONG. IT TESTS THESE 3 CHARS TO SEE IF THEY CONSTITUTE A     %
  % COMPLETE REPLY; IF NOT, IT WAITS FOR THE REST OF THE REPLY.        %

  ERRCODE := NOERROR;
  FOR I:=1 TO 6 DO DEVPARM(I):=0; REP;
                                % ZERO THE DEVICE DEPENDENT PARAMETERS %
  IF OUTLEN > 0 AND INLEN > 0 THEN
    DEVPARM(1):=BUFSET(OUTBUF);
    DEVPARM(2):=OUTLEN;         % SET BUFFER LENGTH                   %
    SECDEV(LUN);                % SECURE THE DATA LINK                %
    MARKTIME(1,50,1);           % 1 SEC TIMEOUT TO OUTPUT MESSAGE     %
    RSXQIW(WAL,LUN,1,0,STATUS,RRNUL,DEVPARM);
                                % WRITE BUFFER OUTBUF TO LUN          %
    CHECKIO(TIMERRO,OUTERR);    % WAS OUTPUT OF MESSAGE SUCCESSFUL ?  %
    DEVPARM(1):=BUFSET(INBUF);   % SET UP PARAMETERS FOR RECEIVING MESSAGE %
    IF INLEN > 3 THEN
      DEVPARM(2):=3;            % FIRST READ IN THREE BYTES     %
      MARKTIME(1,150,1);        % 3 SEC TIMEOUT TO RECEIVE ANSWER FROM LINK
      RSXQIW(RAL LOR RNE,LUN,1,0,STATUS,RRNUL,DEVPARM);
                                % GET INPUT BUFFER,DO NOT ECHO        %
      CHECKIO(TIMERRI,INERR);   % MESSAGE SUCCESSFULLY RECEIVED ?     %
      IF ERRCODE=NOERROR AND INBUF(3) LAND BIT6 = 0 THEN
        % THERE ARE MORE BYTES TO COME : BIT 6 NOT SET AND NO ERROR YET %
        DEVPARM(1):=BUFSET(INBUF) + 3;
        DEVPARM(2):=INLEN - 3;  % INLEN-3 BYTES STILL TO COME         %
        MARKTIME(1,50,1);
        RSXQIW(RAL LOR RNE,LUN,1,0,STATUS,RRNUL,DEVPARM);
        CHECKIO(TIMERRI,INERR);
      END;
    ELSE                % 3 (OR FEWER (?) ) CHARS EXPECTED IN REPLY   %
      DEVPARM(2) := INLEN;
      MARKTIME(1,150,1);
      RSXQIW(RAL LOR RNE,LUN,1,0,STATUS,RRNUL,DEVPARM);
      CHECKIO(TIMERRI,INERR);
    END;
    RELDEV(LUN);                % RELEASE MEDIA                       %
  ELSE                          % BUFFERS OF WRONG LENGTH             %
    ERRCODE:=BUFLENERR;
  END;
  RETURN(ERRCODE);
```

```
ENDPROC;


%   LOCAL PROCEDURES.                                                    %
%   =================                                                    %

PROC BUFSET ( REF ARRAY BYTE BUFFER) INT;
                                % THIS ROUTINE TAKES A BUFFER ADDRESS   %
                                % AND RETURNS IT AS AN INTEGER VARIABLE %
                                % MUST BE COMPILED IN SYSTEMS MODE      %
    INT TEMP;

    CODE 10,0;
        MOV     *BUFFER(%5),*TEMP(%5)    ; MOVE ADDRESS IN BUFFER TO TEMP
        INC     *TEMP(%5)                ; POINT TO FIRST DATA BYTE IN BUFFER
    *RTL;
    RETURN(TEMP);
ENDPROC;

PROC CHECKIO (INT TIMERR,IOERR);
                                % PROC CHECKS EVENT FLAGS AFTER I/O WITH%
                                % TIMEOUT AND ERR ABORTS IF I/O OR TIMEOUT %
                                % ERROR.                                 %

    IF STATUS.IOSTLOW =0 THEN   % TIME OUT HAS OCURRED                   %
        FOR I:=1 TO 6 DO DEVPARM(I):=0;         REP;
        RSXQIW(KIL,LUN,1,0,STATUS,RRNUL,DEVPARM);
        ERRCODE:=TIMERR;
    ELSE
        RSXCMK(0,RRNUL);        % NO TIMEOUT SO CANCEL MARKTIME         %
        IF STATUS.IOSTLOW >= 128 THEN   % I/O ERROR HAS OCCURRED         %
            ERRCODE:=IOERR;
        END;
    END;
ENDPROC;
```

```
OPTION (1) CM;                                                    (N-6-1)
TITLE
MEDLNK;

%                                                                       %
%   DESCRIPTION OF MODULE.                                              %
%   ======================                                              %
%                                                                       %
%   THIS MODULE CONTAINS THE PROCEDURES FOR TRANSFERRING DATA           %
%   BETWEEN THE 'MEDCOM' COMMON DATA BASE IN THE PDP 11/23 AND          %
%   THE REMOTE MEDIA SYSTEM. IT CONTAINS THE FOLLOWING PROCS            %
%   AVAILABLE TO OTHER MODULES FOR READING FROM AND WRITING TO          %
%   MEDIA:                                                              %
%               SINGLIN :       INPUT A SINGLE MEDIA CARD OR            %
%                               ANOLOGUE LIST ITEM FROM MEDIA           %
%               BLOCKIN :       INPUT A CONTIGUOUSLY ADDRESSED          %
%                               BLOCK OF MEDIA CARDS OR AN              %
%                               ANALOGUE LIST FROM MEDIA                %
%               WRITE :         WRITE A 16 BIT WORD TO A SINGLE         %
%                               MEDIA CARD                              %
%               GETMED :        GET THE MEDIA STATUS WORD FROM          %
%                               MEDIA                                   %
%               GETDCW :        GET THE DIGITAL CHANGE WORDS            %
%                               FROM MEDIA                              %
%                                                                       %
%   THESE PROCS ARE NORMALLY ONLY CALLED BY THE MEDIA UPDATE            %
%   TASK.                                                               %
%                                                                       %

%   LET DEFINITIONS.                                                    %
%   ================                                                    %

LET LF  = OCT 012;              % LINE FEED CHARACTER                   %
LET CR = OCT 015;               % CARIAGE RETURN CHARACTER              %
LET VT = OCT 013;               % VERTICAL TAB CHARACTER                %
LET SP = OCT 040;               % SPACE CHARACTER                       %
LET ETX = 3;                    % END OF TEXT                           %
LET ENQ = 5;                    % PROMPT FOR INPUT CHARACTER            %
LET BEL = 7;                    % BELL CHARACTER                        %

LET ERPBASE = 500;              % START OF ERROR NUMBERS                %

LET NOERROR = 0;                % NO ERROR                              %
LET BCCRECERR = 1;              % BCC ERROR IN RECEIVED MESSAGE         %
LET ALIGNERR = 2;               % ALIGNMENT ERR IN REC'D MESG.          %
LET WRONGCODE = 3;              % WRONG CODE IN REC'D MESSAGE           %
LET PARERR = 4;                 % PARITY,ETC ERR IN REC'D MESG.         %
LET BLOCKERR = 5;               % BCC ERR IN TRANSMITTED MSG            %
LET BADCODE = 6;                % INVALID CODE TRANSMITTED              %
LET MEDIAERR = 7;               % MEDIA ERROR                           %
LET LENERR = 8;                 % RECEIVED ANSWER IS WRONG LENGTH       %
LET OUTOFSCAN = 9;              % MEDCARD IS OUT OF SCAN                %
LET WRTERR = 10;                % REC'D MSG NOT IDENTICAL TO            %
                                % XMITTED MSG IN WRITE.                 %
LET ADDRERR = 11;               % BLOCK READ ADDRESSES NOT CONTIGUOUS   %
LET MIXEDADDRS = 12;            % BLOCK READ ADDRESSES MIXED LIST AND   %
                                % MEDIA.                                %
LET TIMEERR = 13;               % TIMEOUT IN XMITTED MESSAGE            %
LET ERRERR = 14;                % INVALID ERROR CODE RETURNED BY MEDIA  %
LET BADBLKLIMS = 15;            % INVALID BLOCK LIMITS ON BLOCKIN()     %

LET RSNGMED = 1;                % CODE : READ SINGLE MEDIA ADDR         %
```

```
LET RSNGANA = 2;                    % CODE : READ SINGLE LIST ADDR  %
LET RBLKMED = 3;                    % CODE : READ BLOCK MEDIA ADDRS %
LET RBLKANA = 4;                    % CODE : READ BLOCK LIST ADDRS  %
LET RDIGCHG = 7;                    % CODE : READ DIG CHG WORDS     %
LET RSNGSTA = 6;                    % CODE : READ MEDIA STATUS WORD %
LET WSNGMED = 12;                   % CODE : WRITE TO MEDIA ADDR    %
LET WSNGANA = 13;                   % CODE : WRITE TO LIST ADDR     %
LET GOMES = 14;                     % CODE : 2ND STAGE OF WRITE     %

LET YES = 1;
LET NO = 0;


%   MODE DEFINITIONS.                                               %
%   =================                                               %

MODE MEDCARD ( INT STAT,MEDDAT,ADDR,REAL SCANTIME);% MEDIA CARD RECORD %
MODE AOREC ( INT UIC,USERINT, ANIN,REAL SETP); % AO RECORD           %

%   EXTERNAL PROCEDURE DEFINITIONS.                                 %
%   ===============================                                 %

% THE FOLLOWING ROUTINES ARE IN THE BASE PROGRAMS                   %
EXT PROC () RRNUL;                  % NULL PROCEDURE                %
EXT PROC ( INT ) RRGEL;             % UNRECOVERABLE ERROR HANDLER   %

%   THE FOLLOWING ARE ROUTINES FROM LB:(1,1)MTSLIB.OLB              %
EXT PROC ( INT ) DELAY;             % DELAY TASK BY NNN TICKS       %
EXT PROC ( BYTE ) BYTE ODDPAR;      % SET ODD PARITY ON BYTE        %
EXT PROC ( INT ) OWRT;              % WRITE OUT INT AS OCTAL        %
EXT PROC ( REAL ) REAL TIMER;       % REAL TIME DIFFERENCE          %
EXT PROC () TTIO;                   % SET UP TT I/O                 %

%   THE FOLLOWING ARE STANDARD STREAM I/O PROCEDURES - REFER        %
%   TO THE STREAM I/O MANUAL                                        %
EXT PROC(REF ARRAY BYTE) TWRT;
EXT PROC (INT) IWRT;

%   THE FOLLOWING IS FROM MODULE LINKLB.OBJ                         %
EXT PROC ( REF ARRAY BYTE,INT,REF ARRAY BYTE,INT)INT MESSANS;
                % SEND MESSAGE DOWN DATA LINK AND GET ANSWER.       %
                % RETURN ERROR STATUS.                              %

%    THE FOLLOWING ARE FROM SECREL.RTL                              %
EXT PROC () SECMEDCOM, RELMEDCOM;       % SECURE & RELEASE DATA BASE %

%   SVC DATA BRICK DEFINITIONS.                                     %
%   ===========================                                     %

%   SVC DATA BRICKS ARE PART OF THE INTERFACE TO THE OPERATING      %
%   SYSTEM                                                          %

SVC DATA RRSIO;
      PROC()BYTE IN;                % STREAM I/O INPUT PROCEDURE    %
      PROC(BYTE) OUT;               % STREAM I/O OUTPUT PROCEDURE   %
ENDDATA;

SVC DATA RRSED;
      BYTE TERMCH,
           IOFLAG;
ENDDATA;
```

```
SVC DATA RRERR;
      LABEL ERL;
      INT ERN;
      PROC(INT) ERP;
ENDDATA;

%   EXTERNAL DATA BRICK DEFINITIONS.                                    %
%   ================================                                    %

%   LOCAL DATA BRICK DEFINITIONS.                                       %
%   =============================                                       %

DATA LOCAL;
    REF ARRAY BYTE TERMB:="#CR,LF#";     % TERMINATING CHARACTERS FOR INPUT %
    ARRAY (16) BYTE OUTMSG;              % MESSAGES TO MEDIA OUTPUT BUFFER  %
    ARRAY (80) BYTE INMSG;               % MESSAGES FROM MEDIA INPUT BUFFER %
    INT SAVEADDR;                        % HOLDS PREVIOUS ADDRESS FOR GOMES %
ENDDATA;

%   OTHER ENT PROCEDURES.                                               %
%   =====================                                               %


%**************  MEDIA ACCESS ROUTINES ***************************%

ENT PROC SINGLIN ( INT MCODE,REF MEDCARD INP) INT;      % PROC TO READ A %
                        % SINGLE INPUT FROM MEDIA. RESULT IS 16 BIT INT%
                        % THE ERROR STATUS IS RETURNED.                 %

    INT RET;

    RET:=READ(MCODE,INP.ADDR,1);         % READ SINGLE MEDIA OR LIST ADDRESS
    IF RET = NOERROR THEN                % NO FAULT IN READ              %
      SECMEDCOM();                       % SECURE DATA BASE
%
      INP.MEDDAT:=DECODE(1);     % GET DECODED DATA FROM INPUT BUFFER    %
      INP.SCANTIME:=TIMER(0.0); % GET REAL TIME OF SCAN                  %
      RELMEDCOM();               % RELEASE DATA BASE                     %
    END;
    RETURN(RET);
ENDPROC;

ENT PROC BLOCKIN ( INT MCODE,FIRST,LAST,REF ARRAY MEDCARD MBLOCK ) INT;
                                % PROC TO READ IN A BLOCK OF            %
                                % CONSECUTIVE MEDIA CARDS OR AN ANALOG  %
                                % MULTIPLEXED LIST FROM MEDIA MEMORY    %
% READS IN LAST-FIRST + 1 ITEMS AND PUTS THE VALUES INTO ELEMENTS FIRST %
% TO LAST OF ARRAY MBLOCK.                                              %
% LAST MUST BE GREATER THAN FIRST.                                      %

  INT RET:=NOERROR;

  IF MCODE # RDIGCHG THEN
    RET:=CHECLST(MBLOCK,FIRST,LAST);%CHECK THAT ARRAY IS SUITABLY ORDERED%
  END;
  IF RET = NOERROR THEN          % NO ERROR DETECTED BY CHECLST          %
    RET:=READ(MCODE,MBLOCK(FIRST).ADDR,LAST-FIRST+1);% DATA LINK HANDLING%
    IF RET = NOERROR THEN
     SECMEDCOM();                % SECURE DATA BASE                      %
      FOR I:=FIRST TO LAST DO % GET DECODED DATA INTO DATA BASE %
        MBLOCK(I).MEDDAT:=DECODE(I-FIRST+1);
        MBLOCK(I).SCANTIME:=TIMER(0.0);
      REP;
```

N-6-3

```
  STOPAST();
ENDPROC;

PROC HOME();
  OUT(ESC); OUT('H');
ENDPROC;

PROC CLEOS();
  OUT(ESC); OUT('J');
ENDPROC;

ENT PROC SECMEDCOM();

% THIS PROC DOES NOTHING. IT IS HERE BECAUSE THE ROUTINES IN MEDLNK CALL
%
% A PROC OF THIS NAME TO SECURE MEDCOM. SINCE WE ARE NOT USING MEDCOM, THERE
%
% IS NO NEED TO SECURE IT (!) AND THIS IS HOW WE AVOID DOING IT.
%

ENDPROC;

ENT PROC RELMEDCOM();

% SEE COMMENTS TO SECMEDCOM().
%

ENDPROC;
```

```
% MEDLNKTST -- PROGRAM TO TEST THE ROUTINES IN MEDLNK.RTL        %

LET     LF      =       OCT 012;
LET     ENQ     =       OCT 005;
LET     ETX     =       OCT 003;
LET YES = 1;
LET NO = 0;

MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);

    EXT PROC  (INT,REF MEDCARD)INT SINGLIN,WRITE;
    EXT PROC (INT,INT,INT,REF ARRAY MEDCARD)INT BLOCKIN;
    EXT PROC (REF MEDCARD)INT GETMED;
    EXT PROC (REF ARRAY MEDCARD)INT GETDCW;
    EXT PROC(REF ARRAY BYTE)TWRT;
    EXT PROC()INT IREAD,OREAD;
    EXT PROC(INT) IWRT,OWRT;
    EXT PROC() GPIO;

DATA LOCAL;
    INT ADR;
    INT NITEMS;
    INT DAT;
    INT N;
    INT COMMAND;
    INT ERRNO;
    MEDCARD MCARD:=(1,0,0,0.0);
    ARRAY (8) MEDCARD MBLOCK:=((1,0,0,0.0),
                               (1,0,1,0.0),
                               (1,0,2,0.0),
                               (1,0,3,0.0),
                               (1,0,4,0.0),
                               (1,0,5,0.0),
                               (1,0,6,0.0),
                               (1,0,7,0.0));
    ARRAY(2) MEDCARD DCWS:=    ((1,0,8,0.0),
                               (1,0,9,0.0));
    MEDCARD MSTAT:=(1,0,0,0.0);
ENDDATA;

ENT PROC RRJOB();
  INT BLKSIZE:=0;
  INT STOP :=NO;
  GPIO();

  WHILE STOP=NO DO
    TWRT("#LF#              MEDLNK TEST MENU#LF,LF#");
    TWRT("1.    SINGLE MEDIA READ#LF#");
    TWRT("2.    SINGLE LIST READ#LF#");
    TWRT("3.    BLOCK MEDIA READ#LF#");
    TWRT("4.    BLOCK LIST READ#LF#");
    TWRT("5.    DIGITAL CHANGE WORDS#LF#");
    TWRT("6.    MEDIA STATUS WORD#LF#");
    TWRT("7.    WRITE TO MEDIA#LF#");
    TWRT("8.    WRITE TO LIST#LF#");
    TWRT("9.    EXIT#LF#");
    TWRT("=>#ENQ#");

    N := IREAD();

    IF N=1 OR N=2 THEN  % SINGLE READ   %
        COMMAND := N;
```

N-16-1

```
        ADDRESS();
        MCARD.ADDR := ADR;
        MCARD.MEDDAT := 0;
        MCARD.STAT:=1;
        ERRNO := SINGLIN(COMMAND,MCARD);
        TWRT("#LF#Data read (octal) : "); OWRT(MCARD.MEDDAT);
    ELSEIF N=3 OR N=4 THEN % BLOCK READ %
        COMMAND := N;
        TWRT("#LF#Media/list start address of block (octal) :#ENQ#");
        ADR :=OREAD();

        FOR I:=1 TO 8 DO
          MBLOCK(I).ADDR := ADR + I - 1;
        REP;
        TWRT("#LF#No of items(1 to 8) #ENQ#");
        BLKSIZE:=IREAD();
        ERRNO:=BLOCKIN(COMMAND,1,BLKSIZE,MBLOCK);
        TWRT("#LF#Data read (octal) : ");
        FOR I:=1 TO BLKSIZE DO
          OWRT(MBLOCK(I).MEDDAT); TWRT(" ");
        REP;
    ELSEIF N=5 THEN % READ DCW %
        ERRNO:=GETDCW(DCWS);
        TWRT("#LF#Change words (octal) : ");
        OWRT(DCWS(1).MEDDAT);TWRT(" ");OWRT(DCWS(2).MEDDAT);
    ELSEIF N=6 THEN % READ MEDIA STATUS %
        ERRNO:=GETMED(MSTAT);
        TWRT("#LF#Media status word (octal) : ");OWRT(MSTAT.MEDDAT);
    ELSEIF N=7 OR N=8 THEN  %WRITE %
        COMMAND := N + 5;
        ADDRESS();
        TWRT("#LF#Data(octal)#ENQ#");
        DAT := OREAD();
        MCARD.STAT:=1;
        MCARD.MEDDAT:=DAT;
        MCARD.ADDR:=ADR;
        ERRNO:=WRITE(COMMAND,MCARD);
    ELSE
        STOP:=YES;
    END;

    IF STOP=NO THEN
      TWRT("#LF#Error status returned = "); IWRT(ERRNO); TWRT("#LF#");
    END;
  REP;
ENDPROC;

PROC ADDRESS();
        TWRT("#LF#Media/list address (octal) #ENQ#");
        ADR := OREAD();
ENDPROC;
```

N-17-1

```
% MUTEST -- PROGRAM FOR TESTING MEDUSER INTERFACE                    %

LET LF = OCT 12;
LET ENQ = 5;
LET ETX = 3;
LET YES = 1;
LET NO = 0;

MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);

EXT PROC() GPIO;
EXT PROC(REF ARRAY BYTE)TWRT;
EXT PROC(INT)OWRT,IWRT;
EXT PROC()INT OREAD,IREAD;
EXT PROC(REAL) RWRT;
EXT PROC ()REAL RREAD;
EXT PROC (REAL)REAL TIMER;

EXT PROC (INT,INT,INT) WRMEDOUT;
EXT PROC (INT,INT) WRCOMMINT, SETANINP;
EXT PROC (INT) INT RDCOMMINT, GETANINP;
EXT PROC (INT,INT,INT) INT RDMEDIA;
EXT PROC (INT,REF ARRAY INT) ATTACHED;
EXT PROC (INT,INT,INT,REF MEDCARD)READMEDCARD;
EXT PROC (INT,REAL)SETSETPT;
EXT PROC (INT)REAL GETSETPT;
EXT PROC (INT,INT,INT) REAL RDSCANTIME;

EXT PROC ()HOME,CLEOL,CLEOS;

DATA LOCAL;
  ARRAY (32) INT OUTARRAY;
  INT ADSWITCH;
  INT IOSWITCH;
  INT DAT;
  INT N;
  INT CHAN;
  MEDCARD CARD:=(0,0,0,0.0);
ENDDATA;

ENT PROC RRJOB();
  INT STOP:=NO;
  GPIO();
  HOME();CLEOS();
  WHILE STOP=NO DO
    CLEOL();TWRT("          MEDUSER Menu #LF#");
    CLEOL();TWRT("#LF#");CLEOL();
    CLEOL();TWRT("1. ATTACHED            7. SETANINP#LF#");
    CLEOL();TWRT("2. WRMEDOUT            8. GETANINP#LF#");
    CLEOL();TWRT("3. RDMEDIA             9. SETSETPT#LF#");
    CLEOL();TWRT("4. RDSCANTIME         10. GETSETPT#LF#");
    CLEOL();TWRT("5. WRCOMMINT          11. READMEDCARD#LF#");
    CLEOL();TWRT("6. RDCOMMINT#LF#");
    CLEOL();TWRT("            12. Quit#LF#");
    CLEOL();TWRT("#LF#");
    CLEOL();TWRT("Choice #ENQ#");
    N:=IREAD();
    TWRT("#LF#"); CLEOS(); TWRT("#LF#");
    IF N=1 THEN
      DOATTACHED();
    ELSEIF N=2 THEN
      DOWRMEDOUT();
```

```
    ELSEIF N=3 THEN
      DORDMEDIA();
    ELSEIF N=4 THEN
      DORDSCANTIME();
    ELSEIF N=5 THEN
      DOWRCOMMINT();
    ELSEIF N=6 THEN
      DORDCOMMINT();
    ELSEIF N=7 THEN
      DOSETANINP();
    ELSEIF N=8 THEN
      DOGETANINP();
    ELSEIF N=9 THEN
      DOSETSETPT();
    ELSEIF N=10 THEN
      DOGETSETPT();
    ELSEIF N=11 THEN
      DOREADMEDCARD();
    ELSEIF N=12 THEN
      STOP:=YES;
    END;
    HOME();
  REP;
  CLEOS();
  TWRT("#ETX#");
ENDPROC;

PROC DOATTACHED();
  GETADSW();
  ATTACHED(ADSWITCH,OUTARRAY);
  TWRT("#LF#Attached to channels : #LF#");
  FOR I:=1 TO LENGTH OUTARRAY DO
    IF OUTARRAY(I)#0 THEN
        IWRT(OUTARRAY(I)); TWRT(" ");
    END;
  REP;
ENDPROC;

PROC DOWRMEDOUT();
  GETADSW();
  GETCHAN();
  GETDATA();
  WRMEDOUT(ADSWITCH,CHAN,DAT);
  TWRT("#LF#WRMEDOUT complete");
ENDPROC;

PROC DORDMEDIA();
  GETADSW();
  GETIOSW();
  GETCHAN();
  DAT := RDMEDIA(ADSWITCH,IOSWITCH,CHAN);
  TWRT("#LF#Data read (Octal) = ");OWRT(DAT);
ENDPROC;

PROC DOWRCOMMINT();
  GETCHAN();
  GETDATA();
  WRCOMMINT(CHAN,DAT);
  TWRT("#LF#WRCOMMINT complete");
ENDPROC;

PROC DORDCOMMINT();
```

```
  GETCHAN();
  DAT:=RDCOMMINT(CHAN);
  TWRT("#LF#Data read from common integer (Octal) = ");OWRT(DAT);
ENDPROC;

PROC DORDSCANTIME();
  REAL X;
  GETADSW();
  GETIOSW();
  GETCHAN();
  X:=RDSCANTIME(ADSWITCH,IOSWITCH,CHAN);
  TWRT("#LF#Scantime = ");RWRT(X);
  TWRT("#LF#Seconds ago = ");RWRT(TIMER(X));
ENDPROC;

PROC DOSETSETPT();
  REAL X;
  TWRT("#LF#Analog output number #ENQ#");
  CHAN := IREAD();
  TWRT("#LF#Setpoint (Real) #ENQ#");
  X:=RREAD();
  SETSETPT(CHAN,X);
  TWRT("#LF#SETSETPT complete");
ENDPROC;

PROC DOGETSETPT();
  REAL X;
  TWRT("#LF#Analog output number #ENQ#");
  CHAN := IREAD();
  X:= GETSETPT(CHAN);
  TWRT("#LF#Setpoint = "); RWRT(X);
ENDPROC;

PROC DOSETANINP();
  INT ANO,ANI;
  TWRT("#LF#Analog output number #ENQ#");
  ANO := IREAD();
  TWRT("#LF#Analog input number #ENQ#");
  ANI := IREAD();
  SETANINP(ANO,ANI);
  TWRT("#LF#SETANINP complete");
ENDPROC;

PROC DOGETANINP();
  INT ANI,ANO;
  TWRT("#LF#Analogue output number #ENQ#");
  ANO := IREAD();
  ANI := GETANINP(ANO);
  TWRT("#LF#Corresponding analogue input number = ");IWRT(ANI);
ENDPROC;

PROC DOREADMEDCARD();
  GETADSW();
  GETIOSW();
  TWRT("#LF#Card number #ENQ#");CHAN := IREAD();
  READMEDCARD(ADSWITCH,IOSWITCH,CHAN,CARD);
  TWRT("#LF,LF#Stat      : "); OWRT(CARD.STAT);
  TWRT("#LF#Meddat    : "); OWRT(CARD.MEDDAT);
  TWRT("#LF#Addr      : "); OWRT(CARD.ADDR);
  TWRT("#LF#Scantime  : "); RWRT(CARD.SCANTIME);
ENDPROC;
```

N-17-4

```
PROC GETCHAN();
  TWRT("#LF#Channel number #ENQ#");
  CHAN := IREAD();
ENDPROC;

PROC GETDATA();
  TWRT("#LF#Data (Octal) #ENQ#");
  DAT := OREAD();
ENDPROC;

PROC GETADSW();
  TWRT("#LF#ADSWITCH (0=ANALOG, 1=DIGITAL) #ENQ#");
  ADSWITCH := IREAD();
ENDPROC;

PROC GETIOSW();
  TWRT("#LF#IOSWITCH (0=INPUT, 1=OUTPUT) #ENQ#");
  IOSWITCH := IREAD();
ENDPROC;
```

N-18-1

```
TITLE SIMBGS
   PACKAGE TO ENABLE DIGITAL SIMULATION OF ANALOGUE SYSTEMS.
   IT SERVES AS A CORE WHICH CALLS THREE USER-WRITTEN SUBROUTINES
   WHICH SPECIFY EXACTLY WHAT THE ANALOGUE SYSTEM IS.

;

LET LF=OCT 12;
LET ENQ=5;
LET YES=1;
LET NO=0;

EXT PROC () SIMINIT,     % USER-SUPPLIED ROUTINE CALLED BEFORE MAIN LOOP %
            SIMJOB,      % USER-SUPPLIED ROUTINE CALLED INSIDE MAIN LOOP %
            SIMTIDYUP;   % USER-SUPPLIED ROUTINE CALLED AFTER MAIN LOOP  %
EXT PROC () STARTAST,STOPAST,    % AST ROUTINES  %
            HOME,CLEOS,CLEOL;    % SCREEN CURSOR HANDLING ROUTINES       %
EXT PROC ()INT CTLCYET; % AST ROUTINE   %
EXT PROC (INT) DELAY;
EXT PROC(REAL)REAL TIMER;
EXT PROC ()INT IREAD;
EXT PROC ()RRNUL;
EXT PROC () GPIO;
EXT PROC(REF ARRAY BYTE)TWRT;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT)ERP;
ENDDATA;

ENT DATA SIMDATA;
  INT N,                  % NO OF TIMES ROUND MAIN LOOP   %
      DELAYTICKS;% DELAY IN MAIN LOOP, USED IN REALTIME APPLICATIONS  %
  REAL  DT;      % THE TIME-STEP AROUND LOOP     %
  ARRAY(4) REAL TIME;   % THE PAST 4 SAMPLING TIMES (SECS PAST MIDNIGHT) %
ENDDATA;

DATA LOCAL;
  INT   TIDYUP, % FLAG TO INDICATE IF SIMTIDYUP() IS TO BE CALLED       %
        RESTART,% FLAG TO INDICATE IF PROGRAM MUST RESTART ON COMPLETION %
        STOP,   % FLAG TO STOP MAIN LOOP        %
        REALTIME, % FLAG TO INDICATE REALTIME OR OFFLINE CONTROL        %
        AST;    % FLAG TO INDICATE IF THE AST ROUTINES ARE USED %
  PROC () USERROUTINE:=RRNUL;   % THE USER ROUTINE BEING EXECUTED       %
ENDDATA;

ENT PROC RRJOB();
  RESTART:=YES;
  WHILE RESTART=YES DO
    RESTART:=NO;
    INITIALISEFORLOOP();
    SETDEFAULTS();

    USERROUTINE:=SIMINIT;
    SIMINIT();  % USER SUPPLIED INITIALISATION %

    IF AST=YES THEN STARTAST(); END;
    WHILE STOP=NO DO
      N:=N+1;
      GETDT();
```

```
      USERROUTINE:=SIMJOB;
      SIMJOB(); % USER SUPPLIED MAIN LOOP PROCESSING    %

      DELAY(DELAYTICKS);
      IF AST=YES THEN
        IF CTLCYET()=YES THEN
          PROCESSINTERRUPT();
        END;
      END;
    REP;
    IF AST=YES THEN STOPAST(); END;

    IF TIDYUP=YES THEN
      USERROUTINE:=SIMTIDYUP;
      SIMTIDYUP();      % USER SUPPLIED TIDYUP ROUTINE  %
    END;
  REP;
ENDPROC;

PROC INITIALISEFORLOOP();
  GPIO();
  HOME();CLEOS();
  TWRT("   ANALOG SIMULATION PACKAGE#LF,LF#");
  N:=0;
  TIDYUP:=YES;
  STOP:=NO;
ENDPROC;

PROC SETDEFAULTS();
  DELAYTICKS:=50;       % 1 SECOND DELAY ROUND LOOP    %
  AST:=YES;             % SUPPORT AST INTERRUPTS       %
  REALTIME:=YES;        % REALTIME,NOT OFFLINE         %
ENDPROC;

PROC GETDT();
  % THIS ROUTINE WORKS OUT DT IN A REAL-TIME SITUATION. IT CALCULATES   %
  % AN AVERAGE OVER THE PAST 4 SAMPLING TIMES, WHICH IS OPTIMUM FOR THE %
  % INTEGRATION ALGORITHM USED.                                         %
  % IN OFFLINE SITUATION, THIS PROC DOES NOTHING.                       %

  IF REALTIME=YES THEN
    TIME(4):=TIME(3);
    TIME(3):=TIME(2);
    TIME(2):=TIME(1);
    TIME(1):=TIMER(0.0);
    IF N=2 OR N=3 THEN
      DT:= (TIME(1) - TIME(N))/REAL(N-1);
    ELSEIF N>=4 THEN
      DT:= (TIME(1) + TIME(2) - TIME(3) - TIME(4))/4.0;
    END;
  END;
ENDPROC;

ENT PROC INTEGRATE ( REF ARRAY REAL XDOT, REF REAL X, REAL XO);
  IF N=1 THEN
    % SET TO INITIAL VALUE                        %
    VAL X:=XO;
    XDOT(2):=XDOT(1);
  ELSEIF N=2 THEN
    % USE TRAPEZOIDAL RULE                        %
    VAL X:= XO + (XDOT(1) + XDOT(2))*DT/2.0;
    XDOT(3):=XDOT(2);
```

```
    XDOT(2):=XDOT(1);
  ELSEIF N=3 THEN
    % USE SIMPSON'S RULE                                    %
    VAL X:= XO + (XDOT(1) + 4.0*XDOT(2) + XDOT(3))*DT/6.0;
    XDOT(4):=XDOT(3);
    XDOT(3):=XDOT(2);
    XDOT(2):=XDOT(1);
  ELSE
    % USE ADAMS' 4TH ORDER METHOD                            %
    VAL X:= X + (9.0*XDOT(1)+19.0*XDOT(2)-5.0*XDOT(3)+XDOT(4))*DT/24.0;
    XDOT(4):=XDOT(3);
    XDOT(3):=XDOT(2);
    XDOT(2):=XDOT(1);
  END;
ENDPROC;

ENT PROC HALT();
  % STOP PROCESSING OF MAIN LOOP          %
  IF USERROUTINE=SIMJOB THEN
    STOP:=YES;
  ELSE
    ERP(601);
  END;
ENDPROC;

ENT PROC NOAST();
  % DO NO AST INTERRUPT PROCESSING        %
  IF USERROUTINE=SIMINIT THEN
    AST:=NO;
  ELSE
    ERP(602);
  END;
ENDPROC;

ENT PROC OFFLINE();
  % CONTROL IS OFFLINE, NOT REALTIME      %
  IF USERROUTINE=SIMINIT THEN
    REALTIME:=NO;
    DELAYTICKS:=0;
  ELSE
    ERP(603);
  END;
ENDPROC;

PROC PROCESSINTERRUPT();
  INT CHOICE:=0;

  STOPAST();
  WHILE CHOICE<1 OR CHOICE>5 DO
    HOME();CLEOS();
    TWRT("INTERRUPT MENU#LF,LF#");
    TWRT("1. CONTINUE#LF#");
    TWRT("2. TIDY UP AND RESTART#LF#");
    TWRT("3. RESTART#LF#");
    TWRT("4. TIDY UP AND ABORT#LF#");
    TWRT("5. ABORT#LF,LF#");
    TWRT("?#ENQ#");
    CHOICE:=IREAD();
  REP;
  IF CHOICE=2 THEN
    RESTART:=YES;
    STOP:=YES;
```

```
  ELSEIF CHOICE=3 THEN
    RESTART:=YES;
    STOP:=YES;
    TIDYUP:=NO;
  ELSEIF CHOICE=4 THEN
    STOP:=YES;
  ELSEIF CHOICE=5 THEN
    STOP:=YES;
    TIDYUP:=NO;
  END;
  STARTAST();
ENDPROC;
```

```
TITLE PLOTLIB;

% LIBRARY OF ROUTINES TO ASSIST GRAPHICS USING THE TEKTRONIX 4014 %
% PLOTTING TERMINAL.                                              %

LET FF = HEX 0C;
LET GS = HEX 1D;
LET ESC = HEX 1B;
LET ENQ = HEX 05;
LET ETX = HEX 03;
LET LF = HEX 0A;
LET CR = HEX 0D;
LET EOS = HEX 80;
LET SUB = HEX 1A;

LET HI5BITS = HEX 3E0;
LET LOW5BITS = HEX 1F;
LET HIY = BIN 00100000;
LET LOWY = BIN 01100000;
LET HIX = BIN 00100000;
LET LOWX = BIN 01000000;

SVC DATA RRSIO;
  PROC ()BYTE IN;
  PROC (BYTE) OUT;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC (INT) ERP;
ENDDATA;

EXT PROC (REF ARRAY BYTE, REF ARRAY BYTE)INT TREAD;
EXT PROC (INT) RRGEL;
EXT PROC (INT) DELAY;

ENT DATA PLOTDATA;
  REAL XOFFSET, XFACTOR, YOFFSET, YFACTOR;
  REAL XMIN := -1.0, XMAX := 1.0,
       YMIN := -1.0, YMAX := 1.0;
  ARRAY (20) BYTE REPLYBUF;
ENDDATA;

ENT PROC SCALE (REAL XMN, XMX, YMN, YMX);

% SCALE SCREEN SIZE TO THE UNITS THE USER WILL USE      %

  IF XMN>=XMX OR YMN>=YMX THEN
    RRGEL(609)
  ELSE
    XOFFSET := XMIN := XMN;
    YOFFSET := YMIN := YMN;
    XMAX := XMX;
    YMAX := YMX;
    XFACTOR := 1024.0/(XMAX-XMIN);
    YFACTOR := 781.0/(YMAX-YMIN);
  END;
ENDPROC;

ENT PROC ISOSCALE(REAL XCENTRE,YCENTRE,RADIUS)REAL;
```

N-19-2

```
% SCALE SCREEN SIZE TO USER'S UNITS, BUT ENSURE EQUAL SCALE      %
% SIZE IN X AND Y DIRECTIONS. (PRESERVES SHAPES OF OBJECTS)      %

  REAL MAJAXIS;
  MAJAXIS:=2.0*RADIUS*1024.0/781.0;
  SCALE(XCENTRE-MAJAXIS/2.0,XCENTRE+MAJAXIS/2.0,YCENTRE-RADIUS,YCENTRE+RADIUS
  RETURN(MAJAXIS);
ENDPROC;

ENT PROC DRAW (REAL XPOS, YPOS);

% DRAW A LINE FROM THE CURRENT SCREEN POSITION TO (XPOS,YPOS).  %

  INT XSCR, YSCR;
  CONVERT (XPOS, YPOS, XSCR, YSCR);
  SCRDRAW (XSCR, YSCR);
ENDPROC;

ENT PROC MOVE (REAL XPOS, YPOS);

% MOVE TO SCREEN POSITION (XPOS,YPOS) WITHOUT DRAWING A LINE.   %

  INT XSCR, YSCR;
  CONVERT (XPOS, YPOS, XSCR, YSCR);
  SCRMOVE (XSCR, YSCR);
ENDPROC;

ENT PROC GRAPHMODE();

% ENTER GRAPHICS MODE                    %

  OUT(GS);
  SCRDRAW(512, 390);
ENDPROC;

ENT PROC ALPHAMODE();

% ENTER ALPHANUMERIC (TEXT) MODE                 %

  OUT(CR);
ENDPROC;

ENT PROC CROSSHAIRS();

% GENERATE CROSS-HAIRS ON SCREEN         %

  OUT(ESC);
  OUT(SUB);
  OUT(ETX);
ENDPROC;

ENT PROC GETCROSSHAIRS(REF REAL XPOS, YPOS);

% READ THE (X,Y) POSITION OF THE CROSS-HAIRS     %

  INT REPLYLENGTH, XSCR, YSCR;
  OUT(GS);
  CROSSHAIRS();
  REPLYLENGTH := TREAD(REPLYBUF, "#CR, LF#");
  IF REPLYLENGTH#5 THEN
    ERP(610);
    VAL XPOS := (XMAX-XMIN)/2.0;
```

```
    VAL YPOS := (YMAX-YMIN)/2.0;
  ELSE
    XSCR := ((INT(REPLYBUF(2)) LAND LOW5BITS) SLL 5) +
            (INT(REPLYBUF(3)) LAND LOW5BITS);
    YSCR := ((INT(REPLYBUF(4)) LAND LOW5BITS) SLL 5) +
            (INT(REPLYBUF(5)) LAND LOW5BITS);
    VAL XPOS := XOFFSET + ( REAL(XSCR)/XFACTOR );
    VAL YPOS := YOFFSET + ( REAL(YSCR)/YFACTOR );
  END;
ENDPROC;

ENT PROC CLEARSCREEN();
  OUT(ETX);
  OUT(ESC);
  OUT(FF);
  OUT(ETX);
  DELAY(50);
ENDPROC;

PROC SCRMOVE (INT XPOS, YPOS);

% MOVE TO (X,Y) IN SCREEN UNITS (NOT USER UNITS)        %

  OUT(GS);
  SCRDRAW (XPOS, YPOS);
ENDPROC;

PROC SCRDRAW (INT XPOS, YPOS);

% DRAW LINE TO (X,Y) IN SCREEN UNITS (NOT USER UNITS)   %

  IF XPOS>1023 THEN XPOS := 1023
  ELSEIF XPOS<0 THEN XPOS := 0
  END;
  IF YPOS>780 THEN YPOS := 780
  ELSEIF YPOS<0 THEN YPOS := 0
  END;
  OUT(BYTE(((YPOS LAND HI5BITS) SRL 5) LOR HIY));
  OUT(BYTE((YPOS LAND LOW5BITS) LOR LOWY));
  OUT(BYTE(((XPOS LAND HI5BITS) SRL 5) LOR HIX));
  OUT(BYTE((XPOS LAND LOW5BITS) LOR LOWX));
  OUT(ETX);
ENDPROC;

PROC CONVERT (REAL XPOS, YPOS, REF INT XSCR, YSCR);

% CONVERT USER UNITS TO SCREEN UNITS    %

  IF XPOS<XMIN THEN
    XPOS := XMIN;
    ERP (611);
  ELSEIF XPOS>XMAX THEN
    XPOS := XMAX;
    ERP (611);
  END;
  IF YPOS<YMIN THEN
    YPOS := YMIN;
    ERP (611);
  ELSEIF YPOS>YMAX THEN
    YPOS := YMAX;
    ERP (611);
  END;
```

```
  VAL XSCR := INT ( (XPOS - XOFFSET) * XFACTOR );
  VAL YSCR := INT ( (YPOS - YOFFSET) * YFACTOR );
ENDPROC;
```

N-20-1

```
TITLE STAR;

% DEMONSTRATION PROGRAM FOR THE TEKTRONIX PLOTTING LIBRARY      %
% PLOTLIB.  DRAWS A PATTERN OF LINES ON THE SCREEN.             %

LET ETX=3;
LET ENQ=5;
LET SUB=HEX 1A;
LET ESC=HEX 1B;
LET GS = HEX 1D;
LET LF = 10;

LET FALSE = 0;

MODE IOCL(REF ARRAY BYTE BFR,INT N,DV,PTR,MD,TRM);

EXT PROC()BYTE GPIN,TTIN,HSIN;
EXT PROC(BYTE) GPOUT,HSOUT,TTOUT,OUTF;
EXT PROC(REAL)REAL RSIN,RCOS,REXP;
EXT PROC(REAL) RWRTU;
EXT PROC()INT IREAD;
EXT PROC ()REAL RREAD;
EXT PROC(REF ARRAY BYTE,INT)DBGWRT;
EXT PROC(INT)IWRT;
EXT PROC(REF ARRAY BYTE )TWRT;
EXT PROC(REF ARRAY BYTE,REF ARRAY BYTE)INT TREAD;
EXT PROC (INT) DELAY;
EXT PROC () STARTAST,STOPAST;
EXT PROC ()INT CTLCYET, ASTYET;
EXT PROC (REAL, REAL, REAL, REAL) SCALE;
EXT PROC (REAL, REAL, REAL)REAL ISOSCALE;
EXT PROC (REAL, REAL) DRAW, MOVE;
EXT PROC () GRAPHMODE, ALPHAMODE;
EXT PROC () CLEARSCREEN;

DATA LOCAL;
  ARRAY(132)BYTE BUF,TIBUF,TOBUF,TEKINBUF;
  IOCL TEKOUTCL:=(BUF,0,3,0,0,0);
  IOCL TEKINCL := (TEKINBUF,0,4,0,0,0);
  IOCL TTOUTCL:=(TOBUF,0,1,0,0,0);
  IOCL TTINCL:=(TIBUF,0,2,0,0,0);
  ARRAY(64)REAL X,Y;
ENDDATA;

SVC DATA RRCHAN;
  REF IOCL INCL,OUTCL;
ENDDATA;

SVC DATA RRSIO;
  PROC()BYTE IN; PROC(BYTE)OUT;
ENDDATA;

ENT PROC RRJOB();
  INT N;
  IN := GPIN;
  OUT := HSOUT;
  INCL := TTINCL;
  WHILE 1=1 DO
    OUTCL := TTOUTCL;
    TWRT("#LF#NUMBER OF SIDES (1-64) ?#ETX#");
    N := IREAD();
    SETUPPOINTS(N);
```

N-20-Z

```
    OUTCL := TEKOUTCL;
    ISOSCALE (0.0,0.0,1.01);
    CLEARSCREEN();
    GRAPHMODE();
    PLOTIT(N);
    MOVE(0.0,1.0);
    ALPHAMODE();
  REP;
ENDPROC;

PROC PLOTIT(INT N);

% PLOTS A STAR PATTERN WITH N VERTICES  %

  INT L,K;
  INT NBY2;
  K:=1;
  L:=1;
  MOVE(X(1),Y(1));
  FOR I:=1 TO (N-1) :/ 2 DO
    FOR J:=1 TO N DO
      IF L=K AND J#1 THEN
        K:=(K MOD N) + 1;
        L:=K;
        MOVE(X(L),Y(L));
      END;
      L:=((L+I-1) MOD N) + 1;
      DRAW(X(L),Y(L));
    REP;
  REP;
  IF (N MOD 2)=0 THEN
    NBY2:=N :/ 2;
    TO NBY2 DO
      K:=((K+NBY2-1) MOD N) + 1;
      DRAW(X(K),Y(K));
      K:= (K MOD N ) + 1;
      MOVE(X(K),Y(K));
    REP;
  END;
ENDPROC;

PROC SETUPPOINTS(INT N);

% INITIALISES VERTICES OF THE STAR PATTERN      %

  REAL THETA;
  THETA := 2.0*3.1415926/REAL(N);
  FOR I:=1 TO N DO
    X(I):=RCOS((REAL(I-1))*THETA);
    Y(I):=RSIN((REAL(I-1))*THETA);
  REP;
  OUT(ETX);
ENDPROC;
```

N-21-1

```
TITLE : SCREEN CURSOR POSITIONING ROUTINES;

LET LF = OCT 12;
LET ENQ=5;
LET ETX=3;
LET NL = 10;
LET ESC = OCT 33;
LET SP = ' ';

MODE IOCL (REF ARRAY BYTE BFR, INT N,DV,PTR,MD,TRM);

EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC () BYTE HSIN;
EXT PROC (BYTE) HSOUT;
EXT PROC (INT) TIMDAT;

SVC DATA RRSIO; PROC()BYTE IN; PROC(BYTE)OUT ENDDATA;
SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRCHAN; REF IOCL INCL, OUTCL ENDDATA;

ENT DATA IOLOCAL;
  ARRAY(132)BYTE IBUF,OBUF:=(NL,SP(131));
  IOCL ICL := ( IBUF,0,1,0,1,0);        % HAVE SET MD TO 1 FOR BINARY   %
  IOCL OCL := ( OBUF,0,1,1,1,0);        % DITTO          %
ENDDATA;

PROC INITIALISEIO();

% BASED ON TTIO() BUT  USE HSIN,HSOUT FOR BINARY DATA XFER      %

  IN:=HSIN;
  OUT:=HSOUT;
  INCL:=ICL;
  OUTCL:=OCL;
ENDPROC;

ENT PROC GOTOLC(INT LINE,COL);

% PUT CURSOR AT POSITION (LINE,COL) ON SCREEN   %

  IF LINE>0 AND COL>0 THEN
    IF LINE <= 24 AND COL <= 80 THEN
      OUT(ESC);
      OUT('Y');
      OUT(BYTE(LINE + ' ' - 1));
      OUT(BYTE(COL + ' ' - 1));
    ELSE
      ERP(606);
    END;
  ELSE
    ERP(606);
  END;
ENDPROC;

ENT PROC HOME();
  OUT(ESC); OUT('H');
ENDPROC;

ENT PROC CLEOS();
% CLEAR TO END OF SCREEN %
  OUT(ESC); OUT('J');
ENDPROC;
```

N-21-2

```
ENT PROC CLSCREEN();
  HOME();
  CLEOS();
ENDPROC;

ENT PROC CLEOL();
% CLEAR TO END OF LINE  %
  OUT(ESC); OUT('K');
ENDPROC;

ENT PROC PRINTTIME();
  TIMDAT(-1);
ENDPROC;

ENT PROC FORCEBUFFEROUTPUT();
  OUT(ETX);
ENDPROC;
```

```
TITLE AST;                                                    (N-22-1)

% ASYNCHRONOUS SYSTEM TRAP ROUTINES, FOR ASYCH. CHARACTER INPUT.%
% ASYNCH INPUT IS FROM LUN 2, AND THE QIOS USE EVENT FLAG 1.    %

% THE AST ROUTINES PUT THE INCOMING CHARACTER INTO CHAR/LOCAL   %
% AND SET AN EVENT FLAG TO SIGNAL THAT AN AST OCCURRED.  IF THE %
% USER PREFERS NOT TO USE THE ROUTINES THAT WAIT FOR AST WITH   %
% TIMEOUT, HE CAN POLL TO SEE IF AST HAS OCCURRED BY USING      %
% THE ASTYET POLLING ROUTINES.                                  %

LET NOTYET=0;           % NO AST HAS OCCURRED SINCE LAST ONE    %
                        % DEALT WITH.                           %
LET AST=1;              % NON CONTROL-C AST HAS OCCURRED.       %
LET CTLC=2;             % CONTROL-C AST HAS OCCURRED.           %

% Event flag numbers used :                                     %

LET ASTEF=4;            % MUST BE IN RANGE 1..16                %
LET AST2EF=3;           % MUST BE IN RANGE 1..16                %
LET ASTMASK= OCT 000015;        % EF 3 OR 4 OR 1 %
LET CTLCMASK=OCT 000005;        % EF 3 OR 1 %
LET NCTLCMASK=OCT 000011;       % EF 4 OR 1 %

LET YES=1;
LET NO=0;

LET LF=OCT 12;
LET ENQ=5;
LET ETX=3;

MODE IOCL(REF ARRAY BYTE BFR,INT N,DV,PTR,MD,TRM);
MODE IOXD(INT IOST1,IOST2,BITS,TMO);

EXT PROC (INT,INT,INT)MARKTIME;
EXT PROC (INT)CANMARK;
EXT PROC (INT)INT TSTEF;
EXT PROC (REAL)REAL TIMER;
EXT PROC () TTIO,RRNUL;
EXT PROC(INT)IWRT,OWRT;
EXT PROC(INT)SET,RESET,WAIT,DELAY,RRGEL;
EXT PROC(REF ARRAY BYTE)TWRT;
EXT PROC(REF ARRAY BYTE,INT)DBGWRT;

EXT PROC(INT,INT)RSXWTL;

SVC DATA RRSIO;
  PROC()BYTE IN;
  PROC(BYTE)OUT;
ENDDATA;

SVC DATA RRCHAN;
  REF IOCL INCL,OUTCL;
ENDDATA;

SVC DATA RRIOX;
  REF IOXD INXD,OUTXD;
ENDDATA;

SVC DATA RRERRX;
 INT LINENO;
 BYTE UEFLAG,ERRLUN;
```

```
 INT RSXDSW;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT)ERP;
ENDDATA;

DATA LOCAL;
 INT ASTFLAG:=NOTYET;
 INT CHAR:=0;
 IOXD MYINXD:=(0,0,0,0);
 IOXD MYOUTXD:=(0,0,0,0);
ENDDATA;

ENT PROC STARTAST();

% This proc attaches to TI: to start AST processing     %

REF IOXD IOSB:=MYOUTXD;
CODE 52,0;
  .LIST MEB
  .GLOBL  $DSW
  .MCALL  QIO$S,QIOSY$,TTSYM$
  MOV   *IOSB(5),R1
  QIO$S #IO.ATA,#2,#1,,R1,,<#ASTIO,,#AST2>
  MOV   $DSW,*RSXDSW/RRERRX(0)
  BCC   *DOK
*RTL; RRGEL(608);
DOK:
ENDPROC;

ENT PROC WAITAST(INT NTIX)INT;
  % Wait for AST to occur (timeout=NTIX ticks)  %
  MARKTIME(1,NTIX,1);
  RSXWTL(0,ASTMASK);
  CANMARK(1);
  RETURN(ASTYET());
ENDPROC;

ENT PROC WAITCTLC(INT NTIX)INT;
% Wait for ^C AST to occur (timeout=NTIX ticks) %
  MARKTIME(1,NTIX,1);
  RSXWTL(0,CTLCMASK);
  CANMARK(1);
  RETURN(CTLCYET());
ENDPROC;

ENT PROC WAITNCTLC(INT NTIX)INT;
% Wait for non-^C AST to occur (timeout=NTIX ticks)     %
  MARKTIME(1,NTIX,1);
  RSXWTL(0,NCTLCMASK);
  CANMARK(1);
  RETURN(NCTLCYET());
ENDPROC;

ENT PROC WTCCCZ(INT NTIX)INT;
  % WAIT FOR CTLC OR CTLZ TIMING OUT AFTER NTIX TICKS   %
  REAL STARTTIME;
  INT STOP,NWAIT,RET;
  NWAIT:=NTIX;
```

N-22-2

```
  STARTTIME:=TIMER(0.0);
  STOP:=NO;
  WHILE STOP=NO DO
    IF WAITAST(NWAIT)=YES THEN
      IF CHAR=3 OR CHAR=26 THEN % CTLC OR CTLZ %
        STOP:=YES;
        RET:=YES;
      ELSE
        NWAIT:=NTIX - INT(50.0*TIMER(STARTTIME));
        IF NWAIT<0 THEN NWAIT:=0; END;
      END;
    ELSE
      STOP:=YES;
      RET:=NO;
    END;
  REP;
  RETURN(RET);
ENDPROC;

PROC ASTIOPROC();
  % AST SERVICE ROUTINE TO SERVICE AST INTERRUPTS. SHOULD       %
  % NEVER BE CALLED EXPLICITLY                                  %
  RRGEL(607);   % PREVENTITIVE MEDICINE                        %

CODE 34,0;
        .LIST   MEB
        .MCALL  ASTX$S,DECL$S,SETF$S
 ASTIO:
        MOV     #*AST,*ASTFLAG/LOCAL    ;BGS
        MOV     (SP)+,*CHAR/LOCAL
        SETF$S  #*ASTEF
        BCS     DONE
        DECL$S
DONE:   ASTX$S
*RTL; RRGEL(607);
ENDPROC;

PROC AST2PROC();
  % AST SERVICE ROUTINE FOR CONTROL-C AST INTERRUPTS. SHOULD    %
  % NEVER BE CALLED EXPLICITLY.                                 %
  RRGEL(607);   % PREVENTITIVE MEDICINE                        %

CODE 34,0;
        .LIST MEB
        .MCALL  ASTX$S,SETF$S,DECL$S
AST2:
        MOV     #*CTLC,*ASTFLAG/LOCAL
        MOV     (SP)+,*CHAR/LOCAL
        SETF$S  #*AST2EF
        BCS     DONE2
        DECL$S
DONE2:  ASTX$S
*RTL; RRGEL(607);
ENDPROC;

ENT PROC STOPAST();
% Detaches from terminal and stops AST processing       %
  REF IOXD IOSB:=MYOUTXD;
  CODE 48,0;
        .LIST   MEB
        .MCALL  QIOW$S
        .GLOBL  $DSW
```

```
        MOV     *IOSB(5),R1
        QIOW$S  #IO.DET,#2,#1,,R1
        MOV     $DSW,*RSXDSW/RRERRX(0)
        BCC     *EX
  *RTL;
  RRGEL(608);
EX:
ENDPROC;

ENT PROC ASTCHAR()BYTE;
% Return most recent asynch. input character    %
  RETURN(BYTE(CHAR LAND HEX FF));
ENDPROC;

ENT PROC ASTYET()INT;
% Proc returns whether or not an AST has occurred since the last one dealt %
% with.                                                                     %
  INT RET;
  IF ASTFLAG#NOTYET THEN
    ASTFLAG:=NOTYET;
    RESET(ASTEF);
    RESET(AST2EF);
    RET:=YES;
  ELSE
    RET:=NO;
  END;
  RETURN(RET);
ENDPROC;

ENT PROC CTLCYET()INT;
% Proc returns whether or not ^C AST has occurred. %
  INT RET;
  IF ASTFLAG=CTLC THEN
    ASTFLAG:=NOTYET;
    RESET(AST2EF);
    RET:=YES;
  ELSE
    RET:=NO;
  END;
  RETURN(RET);
ENDPROC;

ENT PROC NCTLCYET()INT;
% Proc returns whether or not non-^C AST has occurred   %
  INT RET;
  IF ASTFLAG=AST THEN
    ASTFLAG:=NOTYET;
    RESET(ASTEF);
    RET:=YES;
  ELSE
    RET:=NO;
  END;
  RETURN(RET);
ENDPROC;
```

N-22-4

N-23-1

```
TITLE SIMGAS
SIMULATES THE ACTION OF FOUR GAS-COLUMN SETS OF APPARATUS;

% THIS TASK SIMULATES THE OPERATION OF FOUR SETS OF AIR-COLUMN HEATING  %
% APPARATUS.  THIS APPARATUS IS USED IN THE EE476 REAL-TIME COMPUTING   %
% CLASS PROJECT, AND SIMULATION IS DONE TO ENABLE DEVELOPMENT AND       %
% DEBUGGING OF STUDENT PROGRAMS BEFORE THEY ARE USED TO CONTROL THE     %
% REAL THING.  IT ALSO ALLOWS UP TO FIVE USERS (ONE ON THE REAL THING   %
% AND FOUR USING THE SIMULATION) TO BE EXECUTING THEIR TASKS AT ONCE.   %
%                                                                       %
% THE SIMULATION, ALTHOUGH ONLY FIRST-ORDER, IS A CLOSE APPOXIMATION    %
% OF THE BEHAVIOUR OF THE REAL SYSTEM.                                  %
%                                                                       %
% THE FOUR SIMULATED COLUMNS USE THE FOUR SIMULATION ANALOGUE INPUTS    %
% AND OUTPUTS AS FOLLOWS:                                               %
%                                                                       %
%       COLUMN 1 :  ANALOG INP 17 AND OUTP 17                           %
%       COLUMN 2 :  ANALOG INP 18 AND OUTP 18                           %
%       COLUMN 3 :  ANALOG INP 19 AND OUTP 19                           %
%       COLUMN 4 :  ANALOG INP 20 AND OUTP 20                           %

LET     LF              = OCT 12;
LET     BEL             = 7;
LET     ETX             = 3;    % CONTROL-C END-OF-TEXT %
LET     ENQ             = 5;


LET     NMULT           = 16;   % NUMBER OF (MULTIPLEXED) A.I.'S       %
LET     NSIMAI          = 4;    % SIMULATION ANALOG INPUTS              %
LET     TOTALAI         = 20;

LET     NDIGICARD       = 2;    % NUMBER OF DIGITAL INPUT CARDS         %
LET     NSIMDIGICARD    = 2;    % NUMBER OF SIMULATION DIG. INP. CARDS  %
LET     TOTALDIGICARD   = 4;

LET     NAO             = 4;    % NUMBER OF ANALOGUE OUTPUTS            %
LET     NSIMAO          = 4;    % NUMBER OF SIMULATION ANALOGUE OUTPUTS %
LET     TOTALAO         = 8;    % TOTAL NO OF ANALOGUE OUTPUTS          %

LET     NDIGOCARD       = 1;    % NUMBER OF DIGITAL OUTPUT CARDS        %
LET     NSIMDIGOCARD    = 1;    % SIMULATION DIG. OUT. CARD             %
LET     TOTALDIGOCARD   = 2;


MODE MEDCARD(INT STAT,MEDDAT,ADDR,REAL SCANTIME);
MODE AOREC  (INT UIC,USERINT,REF MEDCARD ANIN,REAL SETPOINT);

EXT PROC(INT) DELAY;
EXT PROC() SECMEDCOM,RELMEDCOM,FREEMEDCOM,MCOMINIT;  % IN SECREL.RTL %
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC ()REAL RREAD;
EXT PROC(REAL) RWRT,RWRTU;
EXT PROC (REAL)REAL TIMER;
EXT PROC() GPIO,CLEANUP;
EXT PROC (INT) IWRT,OWRT;

% THE FOLLOWING PROCS ARE IN THE ANALOGUE SIMULATION PACKAGE:           %
EXT PROC (REF ARRAY REAL,REF REAL,REAL)INTEGRATE;
EXT PROC () NOAST,OFFLINE,HALT;

SVC DATA RRSIO;
  PROC() BYTE IN;
```

```
  PROC(BYTE) OUT;
ENDDATA;
```

N-23-2

```
SVC DATA RRSED;
  BYTE TERMCH,IOFLAG;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT) ERP;
ENDDATA;

SVC DATA RRERRX;
  INT LINENO;
  BYTE UEFLAG,ERRLUN;
  INT RSXDSW;            % HOLDS DSW RESULT OF EXECUTIVE CALLS   %
ENDDATA;

% MEDCOM DATA BASE : %

EXT DATA INAREA;
  ARRAY(TOTALAI) MEDCARD ANINP;
  ARRAY(TOTALDIGICARD) MEDCARD DIGINP;
  ARRAY(2) MEDCARD DIGCHAN;
  MEDCARD MEDSTAT;
ENDDATA;

EXT DATA OUTAREA;
  ARRAY (TOTALAO) MEDCARD ANOUTP;
  ARRAY (TOTALDIGOCARD) MEDCARD DIGOUT;
  ARRAY (TOTALAO) AOREC AODESC;
  ARRAY (TOTALDIGOCARD,16) INT DIGUICS;
ENDDATA;

EXT DATA SIMDATA ;      % IN THE SIMULATION PACKAGE     %
 INT N,DELAYTICKS;
 REAL DT;
 ARRAY(4) REAL TIME;
ENDDATA;

DATA LOCAL;
  ARRAY(4) INT AI,AO;
  ARRAY(4) REAL POUT,DTEMP,DTSAVE;
  ARRAY(4,4)REAL DTDOT;
  REAL TAMBIENT,TAU,K,TAUHEAT,TAUCOOL;
  REAL STARTTIME;
  % LOOKUP TABLE FOR INVERSE OF THERMISTOR CALIBRATION CURVE    %
  ARRAY(160) INT LOOKUP :=
        (0    ,108  ,280  ,455  ,633  ,814  ,998  ,1186 ,1376 ,1570 ,
        1768 ,1969 ,2174 ,2382 ,2594 ,2810 ,3030 ,3254 ,3481 ,3713 ,
        3948 ,4188 ,4431 ,4679 ,4930 ,5184 ,5442 ,5704 ,5969 ,6236 ,
        6506 ,6779 ,7053 ,7329 ,7606 ,7884 ,8163 ,8441 ,8719 ,8996 ,
        9272 ,9547 ,9819 ,10089,10357,10622,10883,11141,11396,11647,
        11894,12138,12377,12613,12844,13072,13296,13515,13731,13944,
        14152,14357,14558,14755,14949,15140,15328,15512,15693,15871,
        16046,16218,16387,16554,16718,16880,17039,17195,17350,17501,
        17651,17799,17944,18087,18229,18368,18505,18641,18775,18907,
        19037,19166,19293,19419,19543,19666,19787,19906,20025,20142,
        20257,20372,20485,20597,20707,20817,20952,21033,21139,21244,
        21348,21451,21553,21654,21754,21853,21952,22049,22145,22241,
        22336,22430,22523,22615,22700,22800,22900,23000,23065,23100,
```

```
        23200,23300,23400,23497,23600,23700,23800,23900,23913,23995,
        24100,24200,24300,24314,24400,24500,24600,24700,24701,24800,
        24900,24900,25000,25076,25200,25200,25300,25300,25438,25500);
ENDDATA;

ENT PROC SIMINIT();

  TWRT("#LF#SIMULATION TASK FOR GAS-COLUMN HEATING PROJECT (EE476)#LF,ETX#");

  % SET THE MEDIA SECURE/RELEASE EVENT FLAG (SEE COMMENTS IN SECREL.RTL) %
  MCOMINIT();

  TAUHEAT:=132.5;        % TIME CONST OF COLUMN HEATING UP (SECONDS)      %
  TAUCOOL:=143.0;        % DITTO FOR COOLING                              %
  K:=0.1131;             % STEADY-STATE CONSTANT (DEGREES CENT. PER WATT) %
  TAMBIENT:=19.0;        % AMBIENT TEMPERATURE (DEG. CENT.)               %
  FOR I:=1 TO 4 DO
    DTDOT(I,1):=0.0;
    DTSAVE(I):=0.0;
  REP;
  DELAYTICKS := 10;
  NOAST();
  STARTTIME:=TIMER(0.0);
ENDPROC;

ENT PROC SIMJOB();

    GETPOWERS();
    COMPUTE();
    IF TIMER(STARTTIME) >= 1.0 THEN
      %SAMPLING INTERVAL HAS PASSED     %
      STARTTIME:=TIMER(0.0);
      WRITETOMEDCOM();
    END;

ENDPROC;

ENT PROC SIMTIDYUP();
  TWRT("SIMTIDYUP#LF#");
ENDPROC;

PROC GETPOWERS ();
  REAL NOW :=TIMER(0.0);
  SECMEDCOM();

  %  READ IN THE ANALOG OUTPUT VALUES FROM MEDCOM AND UPDATE THE SCAN-TIMES
%
  FOR I:=1 TO 4 DO
    AO(I) := (ANOUTP(NAO + I).MEDDAT SRL 7) LAND HEX FF;
    ANOUTP(NAO + I).SCANTIME:= NOW;
  REP;
  RELMEDCOM();

  % CONVERT THE VALUES READ TO WATTS OF OUTPUT POWER              %
  FOR I:=1 TO 4 DO
    POUT(I) := REAL(AO(I))*200.0/51.0;
  REP;

ENDPROC;

PROC COMPUTE();

  % THIS PROC PERFORMS THE ACTUAL ANALOGUE SIMULATION AND DETERMINES    %
```

```
  % (BASED ON THE POWER READ IN FROM THE ANALOG OUTPUTS) WHAT COLUMN     %
  % TEMPERATURE MUST BE WRITTEN TO THE MEDCOM ANALOG INPUTS.             %
  % THE WORKING OF THIS PROC IS EASILY UNDERSTOOD BY REFERRING TO THE    %
  % BLOCK-DIAGRAM OF THE SYSTEM.                                         %

  FOR I:=1 TO 4 DO
    INTEGRATE(DTDOT(I), DTEMP(I), 0.0);
    TAU:=IF DTEMP(I)<=DTSAVE(I) THEN TAUCOOL ELSE TAUHEAT END;
    DTSAVE(I):=DTEMP(I);          % SAVE DTEMP FOR GETTING TAU NEXT TIME ROUND

    DTDOT(I,1) := (K*POUT(I) - DTEMP(I))/TAU;

    % CONVERT THE TEMPERATURE IN DEGREES C. TO THE VALUE THAT THE THERMISTOR
    % WOULD HAVE PRODUCED, BY USING THE INVERSE CALIBRATION LOOK-UP TABLE
   %

    AI(I) := INVCALIB(DTEMP(I)+TAMBIENT);
  REP;
ENDPROC;

PROC INVCALIB(REAL TEMP)INT;

  % PROC TO CONVERT REAL TEMPERATURE IN DEGREES CENTIGRADE TO   %
  % A 10-BIT INTEGER VALUE TO BE WRITTEN TO THE ANALOGUE INPUT SO %
  % AS TO SIMULATE THE VALUE THE REAL TEMPERATURE TRANSDUCER    %
  % WOULD HAVE PRODUCED. THE LOOKUP TABLE IS USED TO APPROXIMATE %
  % THE INVERSE OF THE THERMISTOR INTEGER-TO-TEMPERATURE        %
  % CALIBRATION CURVE.                                          %

  INT N; REAL F;

  IF TEMP <= 17.367 THEN RETURN(0);END;
  IF TEMP >=176.0   THEN RETURN(1023);END;

  N:=INTPART(TEMP-16.0);
  %IE 1 TO 160 IE TEMP BETWEEN 16+N AND 17+N %
  F:=TEMP-16.0-REAL(N); %FRACTIONAL PART OF TEMP %

  RETURN( INT((REAL(LOOKUP(N)) + REAL(LOOKUP(N+1) - LOOKUP(N))*F)/25.0) );

  % (VALUES IN LOOKUP TABLE ARE 100 TIMES AS LARGE AS THE 8-BIT VALUES  %
  % FOR WHICH THE THERMISTOR WAS CALIBRATED, SO TO RETURN A 10-BIT VALUE%
  % WE MUST DIVIDE BY 25 RATHER THAN 100 ).                             %
ENDPROC;

PROC INTPART(REAL RNO)INT;
  % RETURNS THE INTEGER PART OF A REAL NUMBER %
  % ONLY VALID IF RNO > 0.0 %
  RETURN(INT(RNO-0.5));
ENDPROC;

PROC WRITETOMEDCOM();
  REAL NOW:=TIMER(0.0);
  SECMEDCOM();
  FOR I:=1 TO 4 DO
    ANINP(NMULT + I).MEDDAT:=AI(I) SLL 5;       % DATA IN BITS 5 TO 14 %
    ANINP(NMULT + I).SCANTIME:=NOW;
  REP;
  RELMEDCOM();
ENDPROC;
```

```
        .TITLE  RSXBA2
        .IDENT  /1.2/
;-----------------------------------------------------------------------
;
;                               R S X B A 2
;
;  SHAREABLE (RE-ENTRANT) PART OF RTL/2 STARTUP (BASE) PROGRAM.
;
;  1.1  27-JAN-81  R.W.DEHNING              -AECI-
;  1.2  19-JAN-82  RWD  SVC DATA RREXS added and R0 save for RRGEL to
;                       do checking on environment.
;       30 SEP 82  RRGLFACS included.    (B.G.Sherlock)
;
;  THIS IS BASICALLY THE VERSION FROM SPL, BUT MODIFIED TO
;  GIVE MORE EXTENSIVE DIAGNOSTICS WHEN THE STACK DUMP VERSION
;  IS USED ( ZERRD DEFINED )
;
;  PROCEDURES, GLOBAL ENTRY POINTS ETC:
;       RSXBA2          - ENTRY POINT FROM RSXBA1
;       RRSIO           - SVC DATA BRICK OFFSETS
;       RRSED
;       RRERR
;       RRERRX
;       RRTASK
;       RREXS
;       RRCHAN
;       RRIOX
;       RRFDB
;       RRSTK
;       RSXEXI          - SHUTDOWN CODE FOR TASK
;       BA3BPT          - LINE NO (BPT) TRACE HANDLER
;  PROC RRIPF () BYTE   - INPUT STREAM FAILURE
;  PROC RROPF (BYTE)    - OUTPUT STREAM FAILURE
;       RRGMEM          - ODD ADDRESS/BUS TIMEOUT
;       RRGMVI          - MMU EXCEPTION TRAP HANDLER
;       RRGIOT          - IOT TRAP HANDLER
;       RRGRES          - ILLEGAL/RESERVED INSTRUCTION TRAP HANDLER
;       RRGEMT          - NON-RSX EMT TRAP HANDLER
;       RRGFPX          - FL.PT. EXCEPTION TRAP HANDLER
;       RRFACS          - FACILITY TABLE OF TASK NAMES
;       RRGLFA(CS)      - FACILITY TABLE OF TASK NAMES (GLOBAL EF'S)
;  PROC CLEANUP ()      - RELEASES ALL FACILITIES HELD BY TASK
;
;  OPTIONS:
;    DEFINE ZERRD FOR STACK & REGISTER DUMP VERSION
;           ZERRS     SST HANDLERS (RSXBA1 MUST BE COMPATIBLE)
;           ZFACS     INCLUDE FACILITY HANDLING TABLE
;
;  THIS MODULE MUST BE ASSEMBLED WITH LB:[1,1]EXEMC.MLB          *1.2*
;-----------------------------------------------------------------------

ZERRS:
ZERRD:
ZFACS:

        .MCALL  HDRDF$
        HDRDF$                      ; DEFINE TASK HEADER OFFSETS

        .PSECT  RSXBA2

        .GLOBL  RRGEL,RRIPF,RROPF,RRNUL,R25
        .GLOBL  BA3BPT,CNTRTN,EX$SUC,RRSTKL
```

```
        .MCALL  GTSK$S,EXST$S,SVTK$S,SETF$S
        .LIST   MEB

;-----------------------------------------------------------------------
; INITIALIZE SVC DATA BRICKS HELD ON STACK
; AND SET SST VECTORS FOR (AT LEAST) 'TRAP' AND 'BPT'
;
;
; SVC OFFSETS FROM BEGINNING OF STACK (ADDR IN R0)
;
;  NOTE:  RRSTK MUST BE ON THE TOP OF THE SVC DATA AREA, SINCE RSXBA2
;         ASSUMES THAT STKLMT IS ALREADY CORRECTLY POSITIONED ON THE
;         STACK ON ENTRY FROM RSXBA1.  HENCE, IF ADDING NEW SVC DATA
;         AREAS, ADD THEM BELOW RRSTK AND ADJUST THE RRSTK VALUE TO
;         SUIT.  *BE CAREFUL*
;-----------------------------------------------------------------------

        RRSIO   == 0
        RRSED   == 4
        RRERR   == 6
        RRERRX  == 16
        RRTASK  == 24
        RREXS   == 30           ; *1.2*
        RRCHAN  == 32           ; *1.2*
        RRIOX   == 36           ; *1.2*
        RRFDB   == 44           ; *1.2*
        RROCP   == 46           ; *1.2*
        RRSTK   == 52           ; *1.2*

        ERL     = 0
        ERN     = 4
        ERP     = 6

        LINENO  = 0
        UEFLAG  = 2
        ERRLUN  = 3
        RSXDSW  = 4

        EOS     = 200
        CR      = 13
        LF      = 10

;-----------------------------------------------------------------------
;  RSXBA2 IS ENTERED HERE FROM RSXBA1, WITH THE FOLLOWING STACKED:
;
;   SP+4 = STKLMT       .. STACK LIMIT FOR SVC DATA RRSTK
;   SP+2 = RRJOB        .. ENTRY PROC FOR USER TASK
;   SP   = SSTTBL       .. PARTICULAR SST FOR HIS TASK
;-----------------------------------------------------------------------

RSXBA2::

;       RESERVE STACK FOR RRGEL/RRERP EXCLUSIVE USE
;       (OTHERWISE STACK OVERFLOW CANNOT BE REPORTED BY R01)

          ADD     #RRSTKL,4(SP)

; SET UP SST VECTOR PASSED FROM RSXBA1

SSTVL   = 8.                    ; LENGTH OF SST VECTOR
```

```
        MOV     (SP),R2         ; R2 POINTS TO SST VECTOR          *1.2*
        SVTK$S  R2,#SSTVL       ; TO PUT ITS ADDRESS IN TASK HDR *1.2*
                                ; (NOTE: R2 MUST REMAIN INTACT)  *1.2*
        MOV     R2,@#H.TKVA     ; KEEP SST VECTOR ADDRESS IN OUR *1.2*
                                ; COPY OF HEADER. RSX DOES ITS COPY
        TST     (SP)+           ; POP THE #SSTTBL WORD

; INITIALISE SVC DATA AREA ON STACK

        MOV     SP,R0           ; POINTS TO SPACE FOR STKLO
                                ; (WHICH PRESENTLY CONTAINS #RRJOB)
        MOV     (R0),R4         ; SAVE RRJOB ADDR TO CALL IT LATER
        SUB     #RRSTK+2,SP     ; RESERVE SPACE FOR SVC DATA AREA
        MOV     SP,(R0)         ; SET STKLO TO DUMMY LINK CELL
        MOV     SP,R5           ; R5 POINTS TO DUMMY LINK CELL

        CLR     -(R0)           ; CLEAR RROCP (G,S)
        CLR     -(R0)

        CLR     -(R0)           ; CLEAR RRFDB

        CLR     -(R0)           ; CLEAR RRIOX (INXD,OUTXD,TF)
        CLR     -(R0)
        CLR     -(R0)

        CLR     -(R0)           ; CLEAR RRCHAN (INCL,OUTCL)
        CLR     -(R0)

        MOV     #EX$SUC,-(R0)   ; EXS = SUCCESS (=1)               *1.2*

; GET TASK NAME INTO SVC DATA BRICK RRTASK

        SUB     #32.,SP         ; RESERVE SPACE FOR GTSK$S
        MOV     SP,R1           ; POINTER TO TASK NAME SLOT
        GTSK$S  R1
        MOV     2(R1),-(R0)     ; = TASK NAME 2
        MOV     (R1),-(R0)      ; = TASK NAME 1
        MOV     R5,SP           ; RECOVER STACK SPACE

        CLR     -(R0)           ; RSXDSW := 0
        MOV     #400,-(R0)      ; UEFLAG := 0
                                ; ERRLUN := 1
        CLR     -(R0)           ; LINENO := 0

        MOV     #RRERP,-(R0)    ; ERP    := RRERP
        CLR     -(R0)           ; ERN    := 0
        MOV     SP,-(R0)        ; ERL REG. 5 (LINK CELL)
        MOV     #RSXEXI,-(R0)   ; ERL REG. 7 (ADDRESS)

        MOV     #EOS,-(R0)      ; IOFLAG := 0
                                ; TERMCH := EOS

        MOV     #RROPF,-(R0)    ; OUT    := RROPF
        MOV     #RRIPF,-(R0)    ; IN     := RRIPF

;  AT THIS POINT R0 IS READY FOR RTL/2 USE
;  I.E. IT POINTS AT BASE OF SVC DATA AREA,
;       WHICH IS (CORRECTLY) RRSIO
;  SAVE IT BEHIND SST VECTOR IN RSXBA1 (WHICH WE CAN FIND VIA   *1.2*
;  THE TASK HEADER, SO THAT RRGEL CAN CHECK IF R0 GETS CORRUPTED
;  BY A FAILURE INSIDE AN FCS CALL, FOR EXAMPLE.
```

```
        MOV     R0,-(R2)        ; SAVE R0 FOR RRGEL CHECK       *1.2*

;  CREATE DUMMY HALF LINK CELL ON BASE OF DYNAMIC PART OF STACK,
;  THUS COMPLETING THE SETTING UP OF THE RTL/2 ENVIRONMENT
;  BEFORE CALLING 'RRJOB' AT THE ADDRESS PASSED FROM RSXBA1.

        MOV     SP,(SP)         ; CREATE DUMMY HALF LINK CELL
        JSR     R1,(R4)         ; ENTER RRJOB
        .PAGE
;*****************************************************************;
;                                                                 ;
;  AT THIS POINT, THE USER'S PROGRAM IS IN CONTROL.  IT CAN USE THE ;
;  VARIOUS RTL/2 CALLABLE PROCS IN THIS MODULE, BUT ESSENTIALLY THE ;
;  NEXT TIME THIS MODULE TAKES CONTROL WILL BE WHEN:              ;
;                                                                 ;
;  RRIPF OR RROPF IS CALLED BECAUSE AN ATTEMPT IS MADE TO USE A   ;
;                 STREAM WHICH WAS CLOSED BY AN END OF FILE, OR IN ;
;                 FACT NEVER OPENED.                              ;
;  RRERP          IS CALLED DUE TO A RECOVERABLE ERROR (ERP)      ;
;  RRGEL          IS CALLED BECAUSE OF AN UNRECOVERABLE ERROR, EITHER ;
;                 EXPLICITLY BY THE USER'S TASK, OR BEHIND HIS BACK ;
;                 BY ONE OF THE SYSTEM LIBRARY PROCS, CONTROL ROUTINES;
;  GOTO ERL       EXPLICITLY BY THE USER.  SYSTEM LIBRARY PROCS WILL ;
;                 ALWAYS CALL RRGEL, SINCE THIS DOES THE DIAGNOSTIC ;
;                 PRINTOUT BEFORE DOING A 'GOTO ERL'.             ;
;  RETURN FROM RRJOB  IN WHICH CASE EXECUTION RESUMES DIRECTLY BELOW ;
;                 AT LABEL 'RSXEXI' TO CLEANUP ANY SECURED FACILITIES ;
;                 AND DO AN RSX-11M EXIT$S DIRECTIVE.             ;
;                                                                 ;
;  NOTE:  RRIPF & RROPF BOTH CALL RRGEL.  RRGEL DOES A 'GOTO ERL' ;
;         AFTER PRINTING THE ERROR DUMP.  A 'GOTO ERL' COMES IN AT ;
;         RSXEXI BY DEFAULT, TO SHUT DOWN THE TASK, UNLESS THE USER ;
;         HAS REDIRECTED ERL TO A LABEL WHICH IS IN SCOPE IN HIS TASK ;
;  HENCE, ALL PATHS EVENTUALLY END UP IMMEDIATELY AFTER THIS MESSAGE! ;
;                                                                 ;
;*****************************************************************;




;-----------------------------------------------------------------;
;  RSXEXI = RETURN POINT FROM INITIAL ERL, OR RRJOB               ;
;                                                                 ;
;  CLOSE DOWN TASK BY ISSUING EXIT$ DIRECTIVE                     ;
;-----------------------------------------------------------------;

RSXEXI::

.IF DF ZFACS                    ; CLEANUP IF FACILITIES PRESENT
        JSR     R1,CLEANUP      ; RELEASES ALL FACILITIES STILL
                                ; SECURED BY THE CALLING TASK
.ENDC   ;(ZFACS)

        EXST$S  RREXS+0(R0)     ; EXIT WITH STATUS EXS          *1.2*
        .PAGE
;-----------------------------------------------------------------;
;                               B A 3 B P T                       ;
;                                                                 ;
;  THIS IS THE BPT HANDLER USED IF THE MODULE WAS COMPILED        ;
;  WITH A 'OPTION TR' TO GIVE A SOURCE LINE NUMBER TRACE.         ;
;-----------------------------------------------------------------;
```

```
BA3BPT::
        MOV     @(SP),RRERRX+LINENO(R0) ; BPT ROUTINE
        ADD     #2,(SP)
        RTI



;-------------------------------------------------------------------------
;                R R I P F     &     R R O P F                           ;
;                                                                        ;
;  RRIPF = INPUT FAILURE,  'IN' STREAM NOT SET UP                        ;
;  RROPF = OUTPUT   "      'OUT' STREAM NOT SET UP                       ;
;-------------------------------------------------------------------------

RRIPF::
        TRAP    1               ; DEFAULT FOR IN
        .WORD   2,-4
        MOV     #98.,-(SP)      ; ERN = 98
        BR      RRGCOM          ; GO CALL RRGEL TO CRATER IT

RROPF::
        TRAP    1               ; DEFAULT FOR OUT
        .WORD   2,-4
        MOV     #99.,-(SP)      ; ERN = 99
        BR      RRGCOM          ; GO CALL RRGEL TO CRATER IT



.IF DF ZERRS
;-------------------------------------------------------------------------
;                        S S T     H A N D L E R S                       ;
;                                                                        ;
;  THESE ARE THE HANDLERS FOR (MAINLY HARDWARE FAILURE) TRAPS            ;
;  THROUGH THE SPECIFIED SST VECTOR SET UP FROM RSXBA1.                  ;
;-------------------------------------------------------------------------

RRGMEM::
        MOV     #10001.,-(SP)           ; ERN
        BR RRGCOM

RRGMVI::
        MOV     #10002.,-(SP)
        BR      RRGCOM

RRGIOT::
        MOV     #10003.,-(SP)
        BR      RRGCOM

RRGRES::
        MOV     #10004.,-(SP)
        BR      RRGCOM

RRGEMT::
        MOV     #10005.,-(SP)
        BR      RRGCOM

RRGFPX::
```

```
        MOV     #10006.,-(SP)
.ENDC   ;(ZERRS)                                        (N-24-6)

RRGCOM: JSR     R1,@#RRGEL              ; CRATER IT


.IF DF ZFACS
;-------------------------------------------------------------------------;
;                                R R F A C S                              ;
;                                                                         ;
;  STORAGE FOR FACILITY TASK NAME LOCKS                                   ;
;  EQUIVALENT TO RTL/2:                                                   ;
;    MODE R5ONAME ( INT R5ON1,  R5ON2 );                                  ;
;    ENT DATA RRFACS;                                                     ;
;       ARRAY(32) R5ONAME FACS;                                           ;
;    ENDDATA;                                                             ;
;-------------------------------------------------------------------------;

RRFACS::
        .WORD   128.            ; SIMULATE ARRAY(32) R5ONAME TASKS
        .REPT   32.
        .WORD   0               ; R5ON1
        .WORD   0               ; R5ON2
        .ENDM

;-------------------------------------------------------------------------;
;                            R R G L F A C S                              ;
;                       B.G. SHERLOCK 30 SEP 82                           ;
;                                                                         ;
;  STORAGE FOR FACILITY TASK NAME LOCKS USING GLOBAL EVENT FLAGS          ;
;  EQUIVALENT TO RTL/2:                                                   ;
;    MODE R5ONAME ( INT R5ON1, R5ON2 );                                   ;
;    ENT DATA RRGLFACS;                                                   ;
;      ARRAY(24) R5ONAME GLFACS;                                          ;
;    ENDDATA;                                                             ;
;-------------------------------------------------------------------------;

RRGLFACS::
        .WORD   96.             ; SIMULATE ARRAY(24) R5ONAME
        .REPT   24.
        .WORD   0               ; R5ON1
        .WORD   0               ; R5ON2
        .ENDM

;-------------------------------------------------------------------------;
;                                C L E A N U P                            ;
;                                                                         ;
;  PROC CLEANUP ()       -- RELEASE FACILITIES SECURED BY THIS TASK.      ;
;-------------------------------------------------------------------------;

CLEANUP::
        MOV     #32.,R2                 ; SETUP SOB COUNTER
        MOV     #RRFACS+2,R3            ; AND ADDRESS POINTER
        MOV     #65.,R4                 ; AND FIRST EVENT FLAG NO
1$:     CMP     (R3),RRTASK+0(R0)       ; CHECK TASK NAME 1
        BNE     2$                      ; NOT SECURED BY THIS TASK
        CMP     2(R3),RRTASK+2(R0)      ; CHECK TASK NAME 2
        BNE     2$                      ; NOT SECURED BY THIS TASK

;  THIS FACILITY WAS SECURED BY THIS TASK, SO FLAG IT RELEASED (0,0)
;  AND SET EVENT FLAG ( 65 - 96 ) TO TELL OTHER TASKS IT'S RELEASED
```

```
        CLR     (R3)                    ; CLEAR TASK NAME 1
        CLR     2(R3)                   ; CLEAR TASK NAME 2
        SETF$S  R4                      ; SET APPROPRIATE EVENT FLAG

2$:     ADD     #4,R3                   ; BUMP TO NEXT TASK NAME
        INC     R4                      ; & CORRESPONDING EVENT FLAG
        SOB     R2,1$                   ; AND LOOP 32 TIMES

; SIMILAR CODE TO RELEASE FACILITIES SECURED USING GLOBAL
; EVENT FLAGS.  (B.G. SHERLOCK)

        MOV     #24.,R2                 ; 24. GLOBAL EVENT FLAGS
        MOV     #RRGLFACS+2,R3
        MOV     #33.,R4                 ; LOWEST EVENT FLAG NO IS 33.
3$:     CMP     (3),RRTASK+0(RD)
        BNE     4$
        CMP     2(R3),RRTASK+2(RD)
        BNE     4$
;
        CLR     (R3)
        CLR     2(R3)
        SETF$S  R4
4$:     ADD     #4,R3
        INC     R4
        SOB     R2,3$                   ; LOOP 24. TIMES
;
        RTS     R1                      ; BEFORE RETURNING

.ENDC   ; (ZFACS)

        .END
```

APPENDIX O

SOFTWARE LISTINGS: LSI-11 MEDIA SYSTEM

Listings of all the software modules which form part of the LSI-11 Media system are given here. The SMT system modules are listed as modified for use in this system. Pages of the listings have a circled page number at the top right-hand side of the page. For example, the 3rd page of the 2nd program in this appendix is numbered "O-2-3".

The following are the programs listed in this appendix:

```
TITLE
        COMMUNICATIONS ROUTINE
        LSI 11 SMT REPLACEMENT OF MICRO-MEDIA
;
OPTION(1) BC,TR;

LET NOERROR     = 0;    % NO ERROR DETECTED YET                              %
LET UNEXP       = 1;    % UNEXPECTED CHARS. NO START BYTE ERROR              %
LET LENERR      = 8;    % MESSAGE LENGTH DOES NOT CORRESPOND TO CODE         %
LET TOOMANY     = 2;    % TOO MANY CHARS ERROR                               %
LET TIMEERR     = 3;    % TIMEOUT WAITING FOR INPUT                          %
LET BADCODE     = 4;    % UNUSED CODE                                        %
LET BLOCKERR    = 5;    % ERROR IN THE BLOCK CHECK CHARACTER                 %
LET MEDIAERR    = 6;    % MEDIA ACCESS ERROR                                 %
LET UNEXPGO     = 7;    % UNEXPECTED GO MESSAGE                              %

LET RSNGMED     = 1;    % READ FROM 1 MEDIA ADDRESS                          %
LET RSNGANA     = 2;    % READ FROM 1 ANALOG LIST ADDRESS                    %
LET RBLKMED     = 3;    % READ FROM A BLOCK OF MEDIA ADDRESSES               %
LET RBLKANA     = 4;    % READ FROM A BLOCK OF LIST ADDRESSES                %
LET RDIGCHG     = 7;    % READ THE DIGITAL CHANGE WORDS                      %
LET RSNGSTA     = 6;    % READ THE MEDIA STATUS WORD                         %
LET WSNGMED     = 12;   % WRITE TO MEDIA ADDRESS                             %
LET WSNGANA     = 13;   % WRITE TO ANALOG ADDRESS                            %
LET GOMES       = 14;   % DO THE WRITE AS SPECIFIED                          %

LET NDIGICHG    = 2;    % NO OF DIGITAL CHANGE WORDS                         %
LET NANAIN      = 16;   % NO OF ANALOG INPUTS                                %
LET NANAOUT     = 4;    % NO OF ANALOG OUTPUTS                               %

LET INEVENT     = 1;    % INPUT CHAR EVENT                                   %
% ALSO ARE PREV AND CTLAEV %
LET TIME        = 100;  % TIMEOUT TIME IN 1/50THS OF SECONDS                 %
LET BIT6        = OCT 100;      % BIT 6 MASK                                 %

LET YES         = 1;
LET NO          = 0;

LET LF          = OCT 012;

EXT PROC(BYTE) OUTBYTE,DEFOUT,OUTTTY;
EXT PROC() BYTE INBYTE;
EXT PROC(INT) WAIT,WAITFOR,RESET,SET,SECURE,RELEASE,IWRT,OWRT;
EXT PROC() LOCK,UNLOCK,HLOCK,HUNLOCK,CLEANUP,SVDINIT;
EXT PROC(INT,INT,LABEL)TWAIT;
EXT PROC(REF ARRAY BYTE)TWRT;

SVC DATA RRSIO;
  PROC() BYTE IN;
  PROC(BYTE) OUT;
ENDDATA;

SVC DATA RRSED;
  BYTE TERMCH, IOFLAG;
ENDDATA;

SVC DATA RRERR;
  LABEL ERL;
  INT ERN;
  PROC(INT) ERP;
ENDDATA;
```

```
SVC DATA STKUSG;
  INT USAGE,LINE;
ENDDATA;


MODE IOAREA(INT EXPECT,FULL,INPT,OUTPT, BYTE LASTCH,
            REF ARRAY BYTE INBUFF);

ENT STACK COMSTK 300;


EXT DATA IODATA;
  ARRAY(512) BYTE IOBUFF;
  ARRAY (80) BYTE TTYBUF;
  IOAREA INAREA,TTYA;
ENDDATA;

EXT DATA TIMEDATA;
  INT NOW,SECSNOW,MINSNOW,TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

EXT DATA MULTIDATA;
  ARRAY(NDIGICHG) INT DIGICHG;
  ARRAY(NANAIN)   INT ANALOGUE;
  ARRAY(OCT 20)   INT INSCAN;
ENDDATA;


DATA LOCAL;
  ARRAY (156) BYTE INMSG;
  ARRAY (156) BYTE REPLY;
  INT INP       :=0;    % COUNTER FOR BYTES INPUT.                 %
  INT OUTP      :=0;    % NO OF CHARS IN OUTPUT REPLY.             %
  INT ERRFLG:=NOERROR;  % INDICATES TYPE OF ERROR.                 %
  INT INCODE    :=0;    % MEDIA CODE OF INPUT MESSAGE.             %
  INT LASTCD    :=0;    % LAST MEDIA CODE -- USED FOR GOMESS.      %
  INT ADDR      :=0;    % MEDIA ADDRESS.                           %
  INT VALUE     :=0;    % VALUE OF DATA TO/FROM MEDIA/LIST.        %
  INT STARTTIME :=0;    % START OF PROCESSING OF PRESENT RECORD.%
  INT TIMEOUT   :=0;    % TIME TO WAIT FOR INPUT EVENT FLAG.       %
  INT STATUS    :=0;    %                                          %
  INT WAITIME   :=0;    %                                          %
ENDDATA;


ENT PROC COMS();

        % THIS PROC IS THE BASE PROCEDURE FOR THE SERIAL I/O     %
        % COMMUNICATION USER TASK.  THE TASK IS ACTIVATED EVERY  %
        % TIME THERE IS AN INPUT BYTE FROM HE SERIAL I/O LINE.   %
        % THE CONSOLE INPUT INTERRUPT CODE SETS THE USER EVENT   %
        % INEVENT AND THE COMS TASK WAITS FOR THIS EVENT.        %
        % WEN INEVENT IS SET THE INPUT BYTE IS PROCESSED AND     %
        % THE TASK WAITS FOR THE NEXT INPUT BYTE.  THE BYTES     %
        % ARE BUILT UP INTO RECORDS AND THE ACTUAL CHECKING OF   %
        % THE INPUT IS ONLY DONE WHEN THERE IS A FULL RECORD.    %
        % THE SUCCESSIVE CHARACTERS OF AN INPUT RECORD MUST      %
        % ARRIVE WITHIN 2 SECONDS OF EACH OTHER.  IF THEY ARE    %
        % DELAYED BY MORE THAN THIS THEN THE RECORD IS REJECTED.%

  INT COMPLETE := NO;   % RECORD INPUT BY BUILD() IS COMPLETE    %
```

```
COMSTART:

  IN:=INBYTE;
  OUT := DEFOUT;          % SWALLOW ALL OUTPUT %
  %%%OUT := OUTTTY; %%%
  %%%TWRT("COMSTART#LF#");%%%
  ERP := MYERP;
  ERL := UNRECOV;         % UNRECOVERABLE ERROR PROCESSING LABEL  %
  INP := 0;               % INITIALISE INPUT COUNTER              %
  LASTCD := 0;

  WHILE 1=1 DO            % FOREVER                               %
    IF INP = 0 THEN       % INPUT RECORD IS EMPTY                 %
      %%%TWRT("WAITING FOR 1ST CHAR#LF#");%%%
      WAIT(INEVENT);      % WAIT FOR FIRST INPUT CHAR.            %
      %%%TWRT("INEVENT OCCURRED#LF#");%%%
    ELSE                  % INPUT RECORD IS PARTIALLY FULL.       %
      TIMEOUT := STARTTIME + TIME - NOW;
      %%%TWRT("TWAITING#LF#");%%%
      TWAIT(INEVENT,TIMEOUT,TIMELABEL);

      %IF INEVENT OCCURRED, THEN        %

        %%%TWRT("INEVENT OCCURRED -- NO TIMEOUT#LF#");%%%
        GOTO ENDIF;

      %ELSE, TIMEOUT OCCURRED, SO SET ERROR FLAG        %

TIMELABEL:
        %%%TWRT("TIMEOUT OCCURRED#LF#");%%%
        ERRFLG := TIMEERR;
        PROCESS();        % PROCESS THE RECORD AND                %
                          % RESET INPUT BYTE COUNTER              %

ENDIF:
      %END IF   %

    END;

    WHILE INAREA.INPT > INAREA.OUTPT DO
    %WHILE THERE ARE BYTES IN THE INPUT BUFFER                    %
      COMPLETE := BUILD();      % GET NEXT BYTE INTO RECORD.      %
      STARTTIME := NOW; % SAVE TIME FOR TIMEOUT CALCULATION %
      IF COMPLETE=YES THEN      % END OF INPUT RECORD REACHED.    %
        %THE RECORD IS COMPLETE SO PROCESS IT                     %
        PROCESS();      % PROCESS THE RECORD AND RESET THE        %
                        % INPUT COUNTER                          %
      END;
    REP;

    HLOCK();
    IF INAREA.INPT <= INAREA.OUTPT THEN
        % STILL NO BYTES IN INPUT BUFFER          %
        RESET(INEVENT); % PREPARE TO WAIT FOR NEXT INPUT CHAR     %
    END;
    HUNLOCK();

  %%%TWRT("REP OF FOREVER LOOP#LF#");%%%
  REP;

% WHAT FOLLOWS IS THE UNRECOVERABLE ERROR PROCESSING.  THIS LABEL IS     %
```

```
% NEVER NORMALLY REACHED BECAUSE OF THE REPEAT FOREVER LOOP ABOVE.      %

UNRECOV:

  %%%TWRT("UNRECOV. ERR. LBL.#LF#");%%%
  %CLEANUP();%
  %SVDINIT();%

  % SEND A REPLY INDICATING MEDIA ERROR %
  REPLY(1) := BYTE((INCODE LAND BIN 00001111) LOR BIT6);
  REPLY(2) := BIN 00000100;
  OUTP := 3;
  PUTBCC();
  ANSWER();

  %%%TWRT("GOING TO COMSTART#LF#");%%%
  GOTO COMSTART;

ENDPROC;


ENT PROC MYERP(INT N);
  %%%TWRT("MYERP **************************#LF#");%%%
ENDPROC;

PROC BUILD() INT;

        % THIS PROC IS CALLED FOR EACH INPUT CHARACTER  %
        % AND PLACES THE INPUT BYTES INTO A RECORD.     %
        % RETURNS ZERO NORMALLY, BUT RETURNS ONE WHEN   %
        % END OF RECORD IS DETECTED (BIT 6 SET).        %

  BYTE CH;
  INT ENDMSG := NO;

  %%%TWRT("BUILD#LF#");%%%
  CH := INBYTE();        % GET THE CHAR  %
  INP := INP + 1;        % INCREMENT INPUT COUNTER       %
  INMSG(INP) := CH;      % PUT CHAR INO INMSG BUFFER     %

  % CHECK FOR START OF INPUT             %

  IF INP = 1 THEN
    % FIRST BYTE OF SOURCE TO OUTSTATION MESSAGE        %
    IF CH LAND BIT6 = 0 THEN    % BIT 6 NOT SET; SOMETHING WRONG        %
      INP:=0;   % RESET INPUT COUNTER, IE IGNORE THE CHAR                %
    END;
  ELSE
    IF CH LAND BIT6 # 0 THEN    % BIT 6 IS SET  %
      ENDMSG := YES;
    ELSE
      IF INP = LENGTH INMSG THEN        % BUFFER ABOUT TO OVERFLOW      %
        ERRFLG:=TOOMANY;
        ENDMSG:=YES;
      END;
    END;
  END;
  RETURN(ENDMSG);
ENDPROC;

PROC PROCESS();
```

```
        % THIS PROC PROCESSES THE RECORDS THAT ARE INPUT.          %
        % THE RECORDS ARE DECODE AND THE REQUIRED OPERATION IS     %
        % PERFORMED BY APPROPRIATE PROCEDURES.                     %
        % THE INPUT RECORD POINTER IS ALWAYS RESET TO ZERO.        %

  %%%TWRT("PROCESS#LF#");%%%
  % SET UP THE 1ST 2 BYTES OF THE REPLY MESSAGE, ENSURING THAT    %
  % BIT 6 OF BYTE 2 IS NOT SET.                                   %
  REPLY(1) := INMSG(1);
  REPLY(2) := INMSG(2) LAND BIN 10111111;

  IF ERRFLG=NOERROR THEN          % NO ERROR DETECTED YET         %
    CHKBCC();   % CHESK THAT THE BCC IS CORRECT                   %
    IF ERRFLG=NOERROR THEN        % STILL NO ERROR DETECTED       %
      % EXTRACT THE CODE - 4 LS BITS OF 1ST CHAR                  %
      INCODE:=INMSG(1) LAND OCT 17;
      %%%TWRT("INCODE= "); IWRT(INCODE); TWRT("#LF#");%%%
      IF      INCODE = RSNGMED THEN
        RDSINGLE();       % SINGLE MEDIA READ      %
      ELSEIF INCODE = RSNGANA THEN
        RDSINGLE();       % SINGLE LIST READ       %
      ELSEIF INCODE = RBLKMED THEN
        RDBLOCK();        % BLOCK MEDIA READ       %
      ELSEIF INCODE = RBLKANA THEN
        RDBLOCK();        % BLOCK LIST READ        %
      ELSEIF INCODE = RDIGCHG THEN
        RDCHANGE();       % READ DIGITAL CHANGE WORDS      %
      ELSEIF INCODE = RSNGSTA THEN
        RDSTATUS();       % READ MEDIA STATUS WORD         %
      ELSEIF INCODE = WSNGMED THEN
        WRSINGLE();       % WRITE TO MEDIA         %
      ELSEIF INCODE = WSNGANA THEN
        WRSINGLE();       % WRITE TO LIST          %
      ELSEIF INCODE = GOMES   THEN
        GOMESS();         % CONFIRM WRITE          %
      ELSE                % THE CODE IS NOT USED   %
        ERRFLG := BADCODE;
      END;
    END;
  END;
  %%%TWRT("MESSAGE RECEIVED : INP :"); IWRT(INP); TWRT("#LF#");%%%
  %%%FOR I:=1 TO INP DO%%%
    %%%OWRT(INMSG(I)); TWRT("#LF#");%%%
  %%%REP;%%%
  %%%TWRT("#LF#");%%%

  LASTCD := INCODE;       % SAVE CODE USED IN CASE NEXT CODE IS GOMESS    %
  ERROR();                % PROCESS ANY ERRORS THAT HAVE OCCURRED         %

  PUTBCC();
  ANSWER();

  INP := 0;       % RESET THE INPUT BYTE COUNTER                          %

ENDPROC;

PROC CHKBCC();

% THIS PROC CHECKS THAT THE BCC OF THE INPUT RECORD IS O.K.       %

  INT TST := 0;
```

```
  FOR I:=1 TO INP-1 DO
    TST := TST NEV INMSG(I);    % EXCLUSIVE OR  %
  REP;

  IF TST LAND OCT 77 # INMSG(INP) LAND OCT 77 THEN
    % BCC IS INCORRECT  %
    %%%TWRT("BCC ERROR *****#LF#");%%%
    ERRFLG := BLOCKERR;
  END;
ENDPROC;

PROC ERROR();

% THIS PROC DEALS WITH ALL ERROR CONDITIONS DETECTED.   %
% IT RESETS ERRFLG AND SETS UP THE STATUS AREA OF THE   %
% 2ND BYTE OF THE REPLY.                                %

  IF     ERRFLG = NOERROR  THEN % NO ERRORS             %
    STATUS := 0;
  ELSEIF ERRFLG = TOOMANY  THEN % OVERRUN ERROR         %
    STATUS := 1;
  ELSEIF ERRFLG = TIMEERR  THEN % TIMEOUT ERROR         %
    STATUS := HEX F;
  ELSEIF ERRFLG = BADCODE  THEN % INVALID CODE          %
    STATUS := 8;
  ELSEIF ERRFLG = BLOCKERR THEN % BCC ERROR             %
    STATUS := 2;
  ELSEIF ERRFLG = MEDIAERR THEN % MEDIA ACCESS ERROR    %
    STATUS := 4;
  ELSEIF ERRFLG = LENERR   THEN % MSG IS WRONG LENGTH   %
    STATUS := 1;
  ELSEIF ERRFLG = UNEXPGO  THEN % UNEXPECTED GO MSG     %
    STATUS := 8;                % INVALID CODE ??       %
  END;

  % PUT STATUS INTO BYTE 2 OF REPLY, EXCEPT IN THE CASE %
  % OF SUCCESSFUL 1ST STAGE OF A WRITE, WHERE NO STATUS %
  % INFO IS OUTPUT.                                     %

  IF ERRFLG = NOERROR THEN
    IF INCODE # WSNGANA AND INCODE # WSNGMED THEN
      REPLY(2) := BYTE((INT(REPLY(2) LAND BIN 10110000)) LOR STATUS);
    END;
  ELSE
    REPLY(2) := BYTE((INT(REPLY(2) LAND BIN 10110000)) LOR STATUS);
  END;

  IF ERRFLG # NOERROR THEN
    %%%TWRT("ERROR :"); IWRT(ERRFLG); TWRT("#LF#");%%%
    OUTP:=3;
    LASTCD := 0;          % TO PREVENT AN UNWANTED GO MSG %
    ERRFLG := NOERROR;    % RESET ERRFLG                  %
  END;
ENDPROC;

PROC RDCHANGE();
  % THIS ROUTINE PLACES THE DIGITAL CHANGE WORDS INTO   %
  % THE REPLY BUFFER.                                   %

  IF INP = 4 THEN         % INPUT MSG THE RIGHT LENGTH  %
    FOR I:=1 TO NDIGICHG DO
      ENCODE(DIGICHG(I),I);
```

```
      DIGICHG(I) := 0;
    REP;
    OUTP := 9;
  ELSE
    ERRFLG := LENERR;
  END;
ENDPROC;

PROC RDSTATUS();
  % THIS PROC READS THE MEDIA STATUS WORD AND PUTS IT    %
  % INTO THE REPLY BUFFER.                               %

  INT TMP;

  IF INP = 4 THEN
    CODE 6,0;
      MOV @#167776,*TMP(5)
    *RTL;

    ENCODE(TMP,1);
    OUTP := 6;
  ELSE
    ERRFLG := LENERR;
  END;
ENDPROC;

PROC RDSINGLE();

  % READS A SINGLE VALUE FROM MEDIA OR LIST AND PLACES IT INTO  %
  % THE REPLY BUFFER.                                           %

  INT IN,AD;

  IF INP=4 THEN
    AD:=GETADD();
    IF INCODE=RSNGMED THEN
      IN := RMEDIA(AD);
    ELSE
      IN := ANALOGUE(AD);
      IF INSCAN(OCT 11)=NO OR INSCAN(OCT 12)=NO THEN
        ERRFLG := MEDIAERR;
      END;
    END;
    IF ERRFLG = NOERROR THEN
      ENCODE(IN,1);
      OUTP:=6;
    END;
  ELSE  % MESSAGE THE WRONG LENGTH      %
    ERRFLG := LENERR;
  END;
ENDPROC;

PROC RDBLOCK();

  % READS A BLOCK OF CONSECUTIVE MEDIA/LIST ADDRESSES AND       %
  % PLACES THEM INTO THE REPLY BUFFER.                          %

  INT IN ,AD, NIN;

  IF INP=5 THEN
    AD:=GETADD();
    NIN:=INMSG(4) LAND OCT 77;  % NO OF ITEMS TO BE READ        %
```

```
    IF INCODE=RBLKMED THEN
      FOR I:=1 TO NIN DO
        IN:=RMEDIA(AD+I-1);
        ENCODE(IN,I);
      REP;
      OUTP := 3*(NIN+1);
    ELSE
      IF INSCAN(OCT 11)=YES AND INSCAN(OCT 12)=YES THEN
        FOR I:=1 TO NIN DO
          IN:=ANALOGUE(AD+I-1);
          ENCODE(IN,I);
        REP;
        OUTP := 3*(NIN+1);
      ELSE
        ERRFLG:=MEDIAERR;
      END;
    END;
  ELSE
    ERRFLG := LENERR;
  END;
ENDPROC;

ENT PROC RMEDIA (INT AD) INT;

  % THIS PROC ACCESSES THE MEDIA.  AD IS THE MEDIA ADDRESS TO   %
  % BE READ.                                                    %

  INT TMP;
  %%%TWRT("RMEDIA -- ADDRESS:"); OWRT(AD); TWRT("#LF#");%%%
  CODE 14,0;
    MOV *AD(5),%1
    ASL %1                ; CONVERT ADDRESS TO WORDS
    ADD #164000,%1        ; ADD IN MEDIA BASE ADDRESS
    MOV (1),*TMP(5)       ; MOVE THE MEDIA VALUE TO TMP
  *RTL;
  %%%TWRT("RMEDIA: RETURNING :"); OWRT(TMP); TWRT("#LF#");%%%
  RETURN(TMP);
ENDPROC;

PROC DECODE () INT;
  % DECODES THE VALUE IN THE INPUT MESSAGE      %

  INT BLD:=TMP:=0;

  BLD := INMSG(4) LAND OCT 77;  % GET HIGH BITS %
  TMP := INMSG(5);              % MIDDLE BITS   %
  TMP := TMP SLL 6;
  BLD := BLD LOR TMP;           % JOIN THEM UP  %
  BLD := BLD LAND OCT 3777;     % STRIP ANY JUNK        %
  TMP := INMSG(6);              % LOW BITS      %
  TMP := TMP SLL 11;
  BLD := BLD LOR TMP;           % WHOLE LOT NOW %
  %%%TWRT("DECODE- RETURNING:"); OWRT(BLD); TWRT("#LF#");%%%
  RETURN(BLD);
ENDPROC;

ENT PROC WMEDIA(INT ADDR, VALUE);

  % THIS PROC WRITES THE VALUE TO THE MEDIA ADDRESS ADDR        %

  %%%TWRT("WMEDIA-MADR,DATA:");OWRT(ADDR);TWRT(",");OWRT(VALUE);%%%
  %%%TWRT("#LF#");%%%
```

```
  CODE 14,0;
    MOV *ADDR(5),%1              ; PUT MEDIA ADDRESS IN REGISTER 1
    ASL %1                       ; WORD ALIGN THE ADDRESS
    ADD #164000,%1               ; ADD IN BASE ADDRESS
    MOV *VALUE(5),(1)            ; WRITE TO MEDIA
  *RTL;

ENDPROC;

PROC WRSINGLE();

  % THIS PROC PREPARES FOR A MEDIA OR LIST WRITE.  THE ACTUAL WRITE IS  %
  % ONLY DONE AFTER THE GO MESSAGE IS RECEIVED.                          %

  IF INP=7 THEN                  % INPUT MUST BE 7 CHARS                %
    FOR I:=2 TO 7 DO
      REPLY(I) := INMSG(I);      % RETURN THE RECEIVED DATA              %
    REP;

    WAITIME := NOW;              % SAVE THE CURRENT TIME FOR TIMEOUT     %
                                 % CHECKS LATER.                         %
    ADDR := GETADD();
    VALUE:= DECODE();
    OUTP := 7;
  ELSE
    ERRFLG := LENERR;
  END;

ENDPROC;

PROC GOMESS();

  % THIS PROC WRITES TO MEDIA OR LIST.  THE WRSINGLE() PROC MUST HAVE   %
  % JUST BEEN CALLED TO INITIALISE THE ADDRESS AND DATA.                 %

  IF INP=4 THEN                  % INPUT MSG MUST BE 4 CHARS LONG       %
    IF LASTCD=WSNGMED THEN       % WRITE TO MEDIA ADDRESS               %
      WMEDIA(ADDR,VALUE);
      OUTP := 3;
    ELSEIF LASTCD=WSNGANA THEN   % WRITE TO LIST ADDRESS                %
      ANALOGUE(ADDR) := VALUE;   % USELESS-- RATHER ACCESS ANALOG O/P'S  %
      OUTP :=3;
    ELSE                  % ERROR - NO CALL TO WRSINGLE HAS BEEN MADE    %
      ERRFLG := UNEXPGO;
    END;
  ELSE
    ERRFLG := LENERR;
  END;

ENDPROC;

PROC GETADD () INT;

  % THIS PROC DECODES THE MEDIA OR LIST ADDRESS FROM THE INPUT RECORD.   %
  % THE RESULT RETURNED IS THE INTEGER VALUE FOUND.                      %

  INT BLD := TMP := 0;

  BLD := INMSG(2) LAND OCT 17;
  TMP := INMSG(3) LAND OCT 77;
  TMP := TMP SLL 4;
```

```
  BLD := BLD LOR TMP;

  %%%TWRT("GETADD-- RETURNING : "); OWRT(BLD); TWRT("#LF#");%%%
  RETURN(BLD);
ENDPROC;

PROC ENCODE (INT BLD,I);

  % THIS PROC ENCODES THE INTEGER BLD AND PLACES IT IN THE OUTPUT      %
  % BUFFER IN THE I'TH POSITION.                                       %

  REPLY(I*3)   := BYTE(BLD LAND OCT 77);        % BUILD FIRST BYTE     %
  REPLY(I*3+1) := BYTE((BLD LAND OCT 3700) SRL 6);
  REPLY(I*3+2) := BYTE((BLD LAND OCT 174000) SRL 11);

ENDPROC;

PROC PUTBCC();

  % THIS PROC PUTS THE BCC AT THE END OF THE REPLY BLOCK.              %

  INT SUM := 0;
  BYTE B;

  FOR I:=1 TO OUTP-1 DO
    SUM := SUM NEV REPLY(I);            % EXCLUSIVE OR  %
  REP;

  B := BYTE(SUM LAND OCT 77);           % CONVERT TO BYTE       %
  B := B LOR BIT6;                      % SET BIT 6             %
  REPLY(OUTP) := B;                     % SAVE IN REPLY BUFFER  %
ENDPROC;

PROC ANSWER();

  % THIS PROC WRITES THE REPLY BLOCK TO THE SERIAL LINE              %

  %%%TWRT("ANSWER -- NO OF CHARS : "); IWRT(OUTP);%%%
  FOR I:=1 TO OUTP DO
    %%%TWRT("#LF#"); OWRT(REPLY(I));%%%
    OUTBYTE(REPLY(I));
  REP;
  %%%TWRT("#LF,LF#");%%%
ENDPROC;
```

```
TITLE
        MULTIPLEXER SCAN ROUTINE
        LSI 1123 SMT REPLACEMENT OF MICRO-MEDIA
;

LET LF          = OCT 12;
LET NDIGICHG    = 2;            % NUMBER OF DIGITAL CHANGE WORDS        %
LET NANAIN      = 16;           % NUMBER OF ANALOGUE INPUTS              %
LET NDIGICARDS  = 2;            % NUMBER OF DIGITAL INPUT CARDS          %
LET MUXADDR     = OCT 10;       % MEDIA ADDRESS OF MULTIPLEXER           %
LET ANINPADDR   = OCT 11;       % MEDIA ADDRESS OF ANALOG INPUT CARD     %
LET DIGINADDR   = 0;            % MEDIA ADDRESS OF FIRST DIGITAL INPUT   %
LET DIGOUTADDR  = 4;            % MEDIA ADDRESS OF DIGITAL OUTPUT CARD   %

LET YES         = 1;
LET NO          = 0;



        % EXTERNAL PROCEDURES %
        % =================== %

EXT PROC (BYTE) OUTTTY,DEFOUT,OUTBYTE;
EXT PROC () BYTE INTTY;
EXT PROC (INT) DELAY;
EXT PROC (INT, INT, LABEL)  TWAIT;
EXT PROC (INT, INT) WMEDIA;             % WRITE TO MEDIA        %
EXT PROC (INT) INT RMEDIA;              % READ FROM MEDIA       %
EXT PROC ()  CLEANUP, SVDINIT;          % SMT PROCEDURES        %
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) OWRT,IWRT;


        % SVC DATA BRICKS %
        % =============== %

SVC DATA RRSIO;
        PROC () BYTE IN;
        PROC (BYTE) OUT;
ENDDATA;

SVC DATA RRSED;
        BYTE TERMCH, IOFLAG;
ENDDATA;

SVC DATA RRERR;
        LABEL ERL;
        INT ERN;
        PROC (INT) ERP;
ENDDATA;

        % EXTERNAL DATA DEFINITIONS %
        % ========================= %

EXT DATA MEDIAERRDATA;
        INT     NERRS;          % NUMBER OF MEDIA ACCESS ERRORS %
ENDDATA;

EXT DATA TIMEDATA;
        INT     NOW,
                SECSNOW,
                MINSNOW,
```

0-2-1

```
                TCOUNT,
                SECS,
                MINS,
                HOURS,
                DAYS,
                MONTHS,
                YEARS;
ENDDATA;

        % ENTRY DATA DEFINITIONS %
        % ====================== %

ENT DATA MULTIDATA;
        ARRAY (NDIGICHG) INT DIGICH;
        ARRAY (NANAIN) INT ANAIN;

        % ELEMENT I OF INSCAN INDICATES WHETHER THE MEDIA CARD AT MEDIA %
        % ADDRESS I-1 IS IN SCAN.                                       %
        ARRAY (OCT 20) INT INSCAN := (YES,      % ADR  0: DIGITAL I/P 1-16 %
                                      YES,      % ADR  1: DIGITAL I/P 17-32
                                      NO,NO,
                                      YES,      % ADR  4: DIGITAL OUTPUTS  %
                                      NO,NO,NO,
                                      YES,      % ADR 10: MULTIPLEXER
                                      YES,      % ADR 11: ANALOG INPUTS
                                      NO,NO,
                                      YES,      % ADR 14: ANALOG OUTPUTS
                                      NO,NO,NO);
ENDDATA;

        % LOCAL DATA DEFINITIONS %
        % ====================== %

DATA LOCAL;
        ARRAY (NDIGICARDS) INT OLDIN, DIGIN := (0(NDIGICARDS));
        INT ANAINCOUNT := 1;    % NUMBER IN RANGE 1 TO NANAIN INDICATING %
                                % WHICH ANALOG INPUT IS SCANNED THE CURRENT
                                % TIME ROUND THE FOREVER LOOP.           %
        INT ADDR;               % HOLDS MEDIA ADDR BEING ACCESSED        %
        INT J;                  % HOLDS MEDIA ADDR OF CARD TESTED FOR    %
                                % COMING BACK INTO SCAN.                 %
ENDDATA;

        % STACK DEFINITION %
        % ================ %

ENT STACK MULTISTK 300;

        % ENTRY PROCEDURES %
        % ================ %

ENT PROC MULTI();

        % THIS PROCEDURE IS THE BASE PROCEDURE FOR THE MULTIPLEXER       %
        % SCAN TASK.                                                     %
        % THE TASK RUNS EVERY 1/6 OF A SECOND.  IT SCANS THE             %
        % MULTIPLEXED DATA AND STORES THE VALUES IN A VECTOR.            %
        % THE NEW VALUES ARE COMPARED WITH THE OLD ONES FROM THE         %
        % PREVIOUS SCAN AND ANY THAT CHANGE HAVE A CORRESPONDING BIT     %
        % SET IN A DIGITAL CHANGE WORD.  THIS WORD MAY BE INTERROGATED   %
        % BY ANOTHER TASK AND CHANGES EASILY DETECTED.                   %
```

```
  J := DIGINADDR;        % INITIALISE THE SCAN TEST POINTER                %
  ANAINCOUNT:=1;         % INITIALISE THE ANALOG INPUT POINTER             %

  INSCAN(1) := YES;              % ADR 0 : DIGITAL INPUT                   %
  INSCAN(2) := YES;              % ADR 1 : DIGITAL INPUT                   %
  INSCAN(3):=INSCAN(4):=NO;
  INSCAN(5):=  YES;              % ADR 4 : DIGITAL OUTPUT                  %
  INSCAN(6):=INSCAN(7):=INSCAN(OCT 10):=NO;
  INSCAN(OCT 11) := YES;         % ADR 10: MULTIPLEXER                     %
  INSCAN(OCT 12):= YES;          % ADR 11: ANALOG INPUTS                   %
  INSCAN(OCT 13):=INSCAN(OCT 14):=NO;
  INSCAN(OCT 15):= YES;          % ADR 14: ANALOG OUTPUTS                  %
  INSCAN(OCT 16):=INSCAN(OCT 17):=INSCAN(OCT 20):=NO;

MULTISTART:
  OUT:=DEFOUT;

  %%%OUT := OUTTTY;%%%
  %%%TWRT("#LF#SMTMULTI -- MULTISTART");%%%

  ERL := UNRECOV;        % UNRECOVERABLE ERROR LABEL                 %


  WHILE 1=1 DO           % FOREVER       %

    %%%TWRT("#LF#SMTMULTI: START OF FOREVER LOOP#LF#");%%%

    FOR I:=1 TO NDIGICARDS DO
      IF INSCAN(I)=YES THEN      % THE CARD IS IN SCAN                     %
        ADDR := I-1;     % SAVE ADDRESS IN CASE MEDIA ERROR OCCURS         %
        OLDIN(I) := DIGIN(I);    % SAVE OLD VALUE OF DATA                  %
        DIGIN(I) := RMEDIA(I-1);% DIG INPUTS ARE AT MEDIA ADDRS 0 & 1      %
        IF DIGIN(I) # OLDIN(I) THEN
          % THE DATA HAS CHANGED, SO SET DIG CHG WORD BIT                  %
          DIGICH(1) := DIGICH(1) LOR (1 SLL (I-1));
        END;
      END;
    REP;

    IF INSCAN(MUXADDR+1)=YES AND INSCAN(ANINPADDR+1)=YES THEN

      % THE MUX AND ANALOG I/P CARDS ARE IN SCAN,SO CAN READ NEXT ONE     %

      ANAINCOUNT:=(ANAINCOUNT MOD NANAIN) + 1;  % POINT TO NEXT INPUT     %
      ANAIN(ANAINCOUNT) := GETANA(ANAINCOUNT-1);% GET THE ANALOG DATA     %
    END;

    CHECKACARD();        % CHECK ONE OF THE CARDS TO SEE IF IT HAS COME    %
                         % BACK INTO SCAN.                                 %
  REP;


    % UNRECOVERABLE ERROR PROCESSING %

UNRECOV:
  %%%TWRT("#LF#SMTMULTI: UNRECOV. LABEL ");%%%
  %%%TWRT("#LF#ADDRESS AT FAULT:"); OWRT(ADDR);%%%

  INSCAN (ADDR+1) := NO;% PUT THE OFFENDING ADDRESS OUT OF SCAN %
%  CLEANUP();%           % GET RID OF ALL ATTACHED FACILITIES    %
%  SVDINIT();%           % RESTORE THE STACKS                    %
```

```
  %%%TWRT("#LF#GOING TO MULTISTART");%%%

  GOTO MULTISTART;      % START AGAIN FROM FRESH                    %
ENDPROC;

PROC CHECKACARD();

  % EACH TIME ROUND THE FOREVER LOOP, WE CHECK A CARD TO SEE     %
  % IF IT IS IN SCAN.  THE PURPOSE OF THIS IS TO KEEP THE        %
  % MATRIX INSCAN UP TO DATE.   IF A CARD WAS OUT OF SCAN BUT    %
  % COMES INTO SCAN AGAIN, THEN INSCAN WILL BE UPDATED WITHIN    %
  % FOUR TIMES ROUND THE FOREVER LOOP.                           %
  % ONLY ONE CARD IS CHECKED EACH TIME, BECAUSE WHEN IT IS OUT %
  % OF SCAN IT CAUSES A BRANCH TO THE UNRECOVERABLE LABEL !!!    %

  IF J=DIGINADDR THEN
    J:=DIGINADDR + 1;
  ELSEIF J = DIGINADDR + 1 THEN
    J:=MUXADDR;
  ELSEIF J = MUXADDR THEN
    J:=ANINPADDR;
  ELSEIF J = ANINPADDR THEN
    J:=DIGINADDR;
  ELSE           % SHOULD NOT OCCUR      %
    J:=DIGINADDR;
  END;

  IF INSCAN(J+1)= NO THEN       % CARD IS RECORDED AS OUT OF SCAN       %
    RMEDIA(J);           % TRY TO ACCESS THE ADDRESS OF THE CARD %

    %  AT THIS POINT, THE PROGRAM WILL BOMB TO UNRECOV LABEL IF THE     %
    %  CARD IS STILL OUT OF SCAN.                                       %

    INSCAN(J+1):=YES;           % IF WE GET THIS FAR THEN THE CARD IS IN %
                                % SCAN AGAIN, SO CORRECT INSCAN MATRIX. %
  END;
ENDPROC;

PROC GETANA(INT AD) INT;

        % THIS PROCEDURE GETS THE NEXT ANALOGUE MULTIPLEXED     %
        % INPUT.                                                %
        % THE MEDIA MULTIPLEX CARD HAS ABOUT A 150 MICRO SEC    %
        % SETTLE TIME.  (TIME BETWEEN RECIEVING THE ADDRESS,    %
        % AND GETTING THE VALUE FROM THE A TO D).               %

  INT  SELECT := 0;
  INT RET := 0;

  IF AD >= 0 AND AD <= NANAIN-1 THEN    % ADRESS IS IN RANGE            %
    SELECT := (HEX 0010) LOR AD;
    ADDR := MUXADDR;    % SAVE ADDRESS IN CASE OF MEDIA ERROR   %

    WMEDIA(MUXADDR,SELECT);     % WRITE TO MUX SELECTING REQD CHANNEL   %

    DELAY(3);                   % WAIT 40 TO 60 MILLISECS - LONG ENOUGH %
                                % TO FRY AN EGG, BUT THAT'S HOW LONG IT %
                                % TAKES FOR THE CARDS TO GET READY !!!  %

    ADDR := ANINPADDR;  % SAVE ADDRESS IN CASE OF MEDIA FAULT          %
    RET := RMEDIA(ANINPADDR);                   % GET ANALOGUE INPUT    %
  ELSE                  % ADDRESS OUT OF RANGE          %
```

```
    RET :=0;
  END;

  %%%TWRT("#LF#SMTMULTI*GETANA -- RETURNING "); OWRT(RET);%%%
  RETURN(RET);
ENDPROC;
```

0-2-5

```
TITLE
        SMT DEVICE DRIVERS
        LSI-11 MICRO MEDIA SYSTEM
;
OPTION (1);

LET NTASKS      = 22;                   %                               %
LET NUSERTASKS  = 16;                   % NTASKS-NSTSK                   %
LET GO          =  0;                   %                               %
LET NOGO        = 64;                   %                               %
LET PWFEV       =-10;                   % POWER FAIL EVENT               %
LET LF          = OCT 012;

MODE DELREC( INT TIMUP, TASK, REF DELREC NXT);

LET NTDV16     =  2;
LET NEVENTS    = 32;
LET EVQLEN     = 16;
LET NFAC       = 32;

EXT DATA TASKDATA;
   INT CURTASK,CURTEX,TASKLOCK,NXTCUR,EVIN,EVOUT;
   DELREC ADEL;
   REF DELREC FRPTR;
   BYTE HIPRI,LOPRI;
   ARRAY (NTASKS) INT TIMEOUT;
   ARRAY (NTASKS) DELREC DEL;
   ARRAY (NTASKS) STACK CELL;
   ARRAY (NTASKS) BYTE EVFAC,STAT,PRIO,WTCHN;
   ARRAY (NTDV16,NEVENTS) INT EVBITS;
   ARRAY (NTASKS) REF BYTE STATADS;
   ARRAY (EVQLEN) INT EVQ;
   ARRAY (NFAC) BYTE FACILITY,FACTOTSK;
   ARRAY (NEVENTS) BYTE EVTOTSK;
ENDDATA;
EXT PROC (INT) WAITFOR,RESET,WAIT,SECURE,RELEASE,SYSSTO,QEV,QEVRRET;
EXT PROC() RETEV,RETFIN,PTORTL;
EXT PROC () HLOCK,HUNLOCK;
EXT PROC() INT SYSRTO;

SVC DATA RRSED;
   BYTE TERMCH,IOFLAG;
ENDDATA;

        % MODE DEFINITIONS %
        % ================ %

MODE IOAREA
        (INT            EXPECT,         % INPUT EXPECTED FLAG           %
                        FULL,           % BUFFER FULL FLAG              %
                        INPT,           % NUMBER OF INPUT BYTES         %
                        OUTPT,          % NUMBER OF PASSED CHARACTERS   %
        BYTE            LASTCH,         % LAST CHARACTER PASSED         %
        REF ARRAY BYTE  INBUFF);        % BUFFER                        %

        % EXTERNAL DATA BRICKS %
        % ==================== %


ENT DATA IODATA;
        ARRAY (512) BYTE IOBUF;
        ARRAY (80) BYTE TTYBUF;
```

```
        IOAREA          INAREA := (1,0,0,0,0,IOBUF);
        IOAREA          TTYA   := (0,0,0,0,0,TTYBUF);
ENDDATA;

        % LOCAL DATA BRICKS %
        % ================= %


DATA TTYDATA;
    INT USAGE := 0;                         % TASK CURRENTLY USING TTY      %
                                            % COMPAIRED WITH CURTEX FOR     %
                                            % ARRAY LOOK UP                 %
    BYTE CHAR;                              % LAST CHAR TYPED               %
ENDDATA;


LET TTYFAC       = -15;
LET KBEV         = -13;
LET PREV         = -12;
LET INEVENT      =   1;                     % INPUT BYTE READY EVENT        %

LET NUL          =   0;
LET ENQ          =   5;
LET EOM          =   3;
LET EOS          = 128;
LET NL           =  10;
LET CR           =  13;
LET OPENSQBR     = OCT 133;
LET CLOSESQBR    = OCT 135;
LET BACKSP       = OCT 10;              % FOR VT 52                          %
LET BACKSLASH    = OCT 134;

LET CTLAEV       = -15;
LET CTLBEV       = -14;

ENT PROC OUTTTY(BYTE C);
   INT D := C;
   IF CURTEX#USAGE THEN
      SECURE(TTYFAC);                       % FIRST CHARACTER -SECURE DEVICE%
      USAGE:=CURTEX;
   END;
   TTYA.EXPECT := 0;
   IF C # EOM AND C # EOS
   THEN                                     % NOT LAST CHAR, OUTPUT        %

      % OUTPUT CHARACTER                                                   %
      IF C = ENQ THEN C := OPENSQBR END;
      IF C = NL THEN OUTTTY(CR) END;
        OUTBYTE(C);                 % OUTPUT THE CHARACTER          %
         IF C # NL
         THEN
            IF D = ENQ THEN                 % KEYBOARD INPUT EXPECTED       %
               TTYA.INPT := TTYA.OUTPT := 0;
               TTYA.EXPECT := 1;
               IOFLAG := 0;
               SYSSTO(6000);                % 2 MINUTE TIMEOUT              %
               WAITFOR(KBEV);               % WAIT FOR INPUT COMPLETE       %
                  TTYA.EXPECT := 0;             % TERMINATES AND RELEASE
               IF SYSRTO() = 0 THEN         % TIMEOUT OCCURRED              %
                  TTYA.INPT := 1;
                  TTYA.INBUFF(1) := EOM;
                  IOFLAG :=-1;
```

0-3-2

```
            END;
          ELSE RETURN;
          END;
        END;
   END;
   USAGE := 0;
   RELEASE(TTYFAC);
ENDPROC;

ENT PROC INTTY()BYTE;
    TTYA.OUTPT := TTYA.OUTPT + 1;
    RETURN ( IF TTYA.INPT # 0 AND TTYA.OUTPT <= TTYA.INPT THEN
                TTYA.INBUFF(TTYA.OUTPT) % SOME VALID CHARACTERS AFTER   %
                                        % AN ENQ                       %
                ELSE EOM
                END);
ENDPROC;

ENT PROC INSERIAL();

        % THIS PROC IS NEVER CALLED EXPLICITLY.  IT IS AN       %
        % H-TASK PROC FOR DEALING WITH THE CONSOL INPUT         %
        % INTERRUPT.  (VECTOR 60).                              %
        % THE INPUT BYTE IS PLACED IN A BUFFER INAREA.INBUFF.   %
        %                                                       %
        % THE INPUT BYTES ARE NOT MODIFIED IN ANY WAY BEFORE    %
        % BEING SAVED.                                          %
        % THE BYTE TRANSMITTED IS ALWAYS SAVED IN INAREA.LASTCH %
        % THE KEYBOARD EVENT (-15) IS SET BY THIS INTERRUPT     %
        % PROCEDURE.                                            %
        % TO INDICATE THAT INPUT HAS BEEN RECIEVED USER EVENT   %
        % NUMBER 1 IS SET.  THIS ALLOWS THE COMS TASK TO WAKE   %
        % UP AND PROCESS THE INPUT BYTE.                        %

    RETURN;                                 % PREVENTITIVE MEDICINE %

        % CONSOLE INTERRUPT CODE         %

    CODE 10,0;
        DOT=.                       ; SAVE THE LOCATION COUNTER
        .ASECT
        .=60                        ; SET THE LOCATION COUNTER TO 60
        CONSINT,340                 ; GENERATE THE INTERRUPT VECTOR
        .PSECT DEVDRV
        .=DOT                       ; REVERT TO PREVIOUS LOCATION COUNTER

CONSINT:
        MOVB    @#177562,*CHAR/TTYDATA  ; GET THE BYTE
        JSR     %1,*PTORTL              ; SWITCH TO H TASK ENVIRONMENT
    *RTL;


            % BYTE EXPECTED %
        IF INAREA.INPT >= LENGTH INAREA.INBUFF THEN
                % BUFFER IS FULL %
            INAREA.FULL := 1;
        ELSE
                % SAVE THE CHARACTER IN THE BUFFER %
            INAREA.INPT := INAREA.INPT+1;
            INAREA.INBUFF(INAREA.INPT) := CHAR;
        END;
    INAREA.LASTCH := CHAR;
```

0-3-3

```
    QEV (KBEV);
    QEVRRET(INEVENT);                    % QUE EVENT AND RETURN FROM     %
                                         % H-TASK ENVIRONMENT            %
ENDPROC;

ENT PROC INBYTE() BYTE;

        % THIS ROUTINE RETURNS THE NEXT BYTE FROM THE SERIAL INPUT LINE.%
        % INPUT BUFFER.                                                 %
        % THE BUFFER CAN ONLY BE RESET WHEN ALL THE BYTES THAT HAVE     %
        % BEEN INPUT HAVE BEEN PASSED OUT BY THIS ROUTINE.  THE BYTES   %
        % MUST BE USED AT A RATE GREATER THAN THEY ARE INPUT.           %
        % THE RESETING OF THE POINTERS IN THE I/O CONTROL AREA          %
        % INAREA.INPT AND INAREA.OUTPT MUST NOT BE INTERUPTED BY THE    %
        % INPUT INTERRUPT.  ASUSPENSION OF INTERRUPTS THUS COVERS THE   %
        % RESETTING OF THESE POINTERS.                                  %
        % IF NO CHARACTERS EXISTS IN THE INPUT BUFFER THEN THE PROC     %
        % SETS IOFLAG AND RETURNS A NUL CHARACTER.                      %

    BYTE        CHR;            % BYTE TO USE AS A TEMPORY STORE      %


    IF INAREA.INPT > 0 THEN     % ARE BYTES IN BUFFER                   %
        INAREA.OUTPT := INAREA.OUTPT + 1;       % INCREMENT OUTPUT COUNT
%
        CHR := INAREA.INBUFF(INAREA.OUTPT);     %  PUT BYTE INTO BUFFER %
        IF INAREA.OUTPT >= INAREA.INPT THEN

        % ALL BYTES HAVE NOW BEEN OUTPUT FROM BUFFER, SO CAN  RESET THE %
        % BUFFER POINTERS.                                              %

            HLOCK();

            IF INAREA.OUTPT >= INAREA.INPT THEN
                % STILL OK TO RESET POINTERS    %
                INAREA.INPT := 0;               % RESET BUFFER POINTERS
%
                INAREA.OUTPT:= 0;
            END;

            HUNLOCK();

        END;
    ELSE                        % INAREA.INPT = 0, IE NO BYTES IN BUFFER
%
        CHR := NUL;
        %IOBUF := 1;%
    END;

    RETURN (CHR);

ENDPROC;

PROC OUTSERIAL();

        % THIS PROC IS NEVER CALLES EXPLICITLY.  IT IS AN H-TASK        %
        % ROUTINE TO DEAL WITH THE CONSOLE OUTPUT INTERRUPTS.   %
        % (VECTOR 64).                                                  %

    RETURN;             % JUST IN CASE          %

    CODE 4,0;
    DOT1=.              ;SAVE THE LOCATION COUNTER
    .ASECT
```

```
    .=64                ;GENERATE CODE AT LOCATION 64
    CONSPR,340          ;THE INTERUPT VECTOR WORDS
    .PSECT DEVDRV
    .=DOT1              ;REVERT TO OLD LOCATION COUNTER
CONSPR:
    JSR  %1,*PTORTL     ;GET INTO RTL ENVIRONMENT
    *RTL;
    QEVRRET(PREV);      % QUEUE PRINTER EVENT AND RETURN FROM H-TASK    %

ENDPROC;

ENT PROC OUTBYTE (BYTE CH);

        % THIS ROUTINE OUTPUT ONE BYTE TO THE CONSOLE                   %
        % SERIAL I/O LINE.                                              %

        % CHECK IF THE PORT IS READY                                    %

    CODE 6,0;
        TSTB    @#177564        ; STATUS WORD XCRSR
        BMI     *SND            ;READY
    *RTL;

    RESET (PREV);

    CODE 6,0;
        TSTB    @#177564
        BMI     *SND            ;READY
    *RTL;

    WAIT (PREV);

SND:

    CODE 12,0;
        MOVB    *CH(5),@#177566 ;PUT BYTE IN OUTPUT
        MOV     #100,@#177564   ;ENABLE INTERRUPT
    *RTL;

ENDPROC;
```

```
TITLE
      SMT OPERATING SYSTEM
      ( WITH NEGATIVE TASK NUMBERS )
      USER TASK DEFINITION AND INITIALISATION (READ/WRITE)
      **** MODULE SMTU1 ****
      SMT 4(18) 13-04-1981;

OPTION (1);

LET NTASKS     = 22;                    %                                  %
LET NUSERTASKS = 16;                    % NTASKS-NSTSK                      %
LET NFREETASKS = 14;                     % NUSERTASKS-USEDTASKS               %
LET GO         =  0;                    %                                  %
LET NOGO       = 64;                    %                                  %
LET PWFEV      =-10;                    % POWER FAIL EVENT                   %

MODE TASKLINK ( REF ARRAY STACK TKS,
                REF ARRAY PROC() TKP,
                REF ARRAY BYTE TKPR,
                              TKST,
                PROC () UINIT,
                        USERPF);

EXT DATA SYSIODATA;
   PROC () BYTE CTLAIN;
   PROC (BYTE) CTLAOUT,ERROUT;
ENDDATA;

EXT PROC () BYTE DEFIN,INTTY;
EXT PROC (BYTE) DEFOUT,OUTTTY;
EXT PROC () RRNUL;

EXT PROC () FBPROC,INSTRUCT,SETFMQ,CLOCK,ERPRIN,MULTI,COMS;

EXT STACK FBSTK,INSTK,SYSTACK,CLKSTK,ERRSTK,MULTISTK,COMSTK;

ENT DATA TKDFN;
   % TASKS -5 TO 0 ARE SYSTEM TASKS                                          %
   %  -5. FALL-BACK TASK - LOWEST PRIO EXCEPT FOR TASK -3 (SEE BELOW)        %
   %  -4. CTLA TASK                                                          %
   %  -3. STARTUP AND 'SET' TASK. MAY BE GIVEN A PRIORITY LOWER              %
   %      THAN TASK -5. IT PERFORMS MOST OF THE STARTUP OPERATIONS           %
   %      BEFORE BECOMING THE TASK WHICH SERVICES THE EVENT QUEUE            %
   %  -2. CLOCK TASK - STIMULATED BY CLOCK INTERRUPT                         %
   %  -1. ERROR PRINT TASK                                                   %
   %   0.  SPARE SYSTEM TASK                                                 %
   %   1. MEDIA MULTIPLEXER SCANNING TASK                                    %
   %   2. MEDIA COMMUNICATIONS TASK                                          %

   ARRAY(NTASKS)STACK TKSTACK:=(FBSTK,INSTK,SYSTACK,CLKSTK,ERRSTK,FBSTK,
                                MULTISTK,COMSTK,FBSTK(NFREETASKS));
   ARRAY(NTASKS)PROC () TKPROC:=(FBPROC,INSTRUCT,SETFMQ,CLOCK,ERPRIN,
                                 RRNUL,MULTI,COMS,RRNUL(NFREETASKS));
   ARRAY(NTASKS) BYTE TKPRIO:=(2,240,1,250,200,0,
                               10,20,0(NFREETASKS)),
                      TKSTAT:=(GO,GO,GO,GO,GO,NOGO,
                               GO,GO,NOGO(NFREETASKS));
ENDDATA;


DATA TKLINK;
```

```
   TASKLINK TKDFNPTR := (TKSTACK,TKPROC,TKPRIO,TKSTAT,USERINITS,UPWFAIL);
ENDDATA;


ENT PROC USERINITS();
   % USER EDITED INITIALISATION PROCEDURE                                  %
   % CALLED BY STARTUP TASK BEFORE INTERRUPTS ARE ENABLED                  %

   CTLAIN:=INTTY;
   CTLAOUT:=ERROUT:=OUTTTY;

L:
   CODE 0,0;
   .ASECT
      .=40                    ; THIS CODE PLANTS A POINTER TO THE
      .WORD *TKLINK           ; USER TASK DEFINITION DATA INTO
   .PSECT SMTU1X
      .=*L                    ; LOCATION '40
                              ; THE STARTUP CODE USES THIS AS A
                              ; REFERENCE VARIABLE OF MODE 'TASKLINK'.
                              ; THE COMPONENTS OF THIS MODE WILL THEN
                              ; PROVIDE ACCESS TO THE DATA HELD IN
                              ; DATA BRICK TKDFN.

                              ; NOTE:
                              ; THIS METHOD OF ACCESSING TKDFN
                              ; ENABLES THE BASIC SMT SYSTEM TO
                              ; BE LINKED SEPARATELY. THE USER
                              ; TASKS(INCLUDING THIS MODULE) CAN
                              ; THEN BE ADDED IN A SUBSEQUENT
                              ; LINKING OPERATION.
  *RTL;

   CODE 12,0;                 ; +++++++++ 6 FOR LSI
      MOV   #100,@#177546     ; CLOCK INTERRUPT ENABLE +++ REMOVE
                              ; THIS LINE FOR LSI-11 UNLESS
                              ; KPV11 OR BDV11 CARDS ARE INSTALLED
                              ; TO ENABLE SOFTWARE CLOCK CONTROL,
                              ; OTHERWISE ERROR 30 WILL ALWAYS
                              ; BE REPORTED.
      MOV   #100,@#177560     ; TTY KBD INTERRUPT ENABLE
   *RTL;
ENDPROC;

ENT PROC UPWFAIL();
   % USER EDITED PROCEDURE WHICH IS CALLED ON POWER FAIL RESTART.          %
   % THERE ARE SEVERAL POWER FAIL RESTART MECHANISMS WHICH CAN BE          %
   % IMPLEMENTED DEPENDING ON THE CODE OF THIS PROCEDURE:-                 %

   %  (1) IF THIS PROCEDURE IS NULL, A POWER FAIL RESTART WILL            %
   %      SIMPLY START ALL USER TASKS AS FOR A NORMAL SYSTEM              %
   %      STARTUP (COLD START FROM ZERO).                                 %

   %  (2) IF THIS PROCEDURE ENDS BY CALLING 'RETEV' , THIS WILL           %
   %      ALLOW ALL TASKS TO CONTINUE FROM THE POINT THEY HAD             %
   %      REACHED WHEN POWER FAIL OCCURRED.                               %

   %  (3) THIS PROCEDURE RUNS IN A LIMITED HTASK ENVIROMENT, AND          %
   %      PROCEDURE QEV MAY BE USED TO SET EVENTS. IT IS NOT PERMITTED    %
   %      TO USE START, AND STOP, BUT EXT DATA TASKDATA MAY BE            %
   %      MANIPULATED DIRECTLY, AS IN CLOCKINT - SMTB1. IT IS THE         %
   %      RESPONSIBILITY OF THE USER TO CHECK WHETHER THIS IS             %
```

```
    %       PERMISSIBLE IN HIS SYSTEM. IT WILL USUALLY BE PREFERABLE      %
    %       TO USE AN STASK WAITING ON PWFEV, THE POWER FAIL EVENT TO     %
    %       TIDY UP.                                                      %

    %  (4)  THERE ARE TWO SYSTEM FEATURES WHICH MAY BE USEFUL ON          %
    %       POWER FAIL/RESTART:-                                          %
    %       (A) PWFLAG IN PWFDATA IS SET NON ZERO AFTER A POWER FAILURE.  %
    %           (NOTE: THE SYSTEM NEVER ZEROS THIS FLAG.)                 %
    %       (B) EVENT NUMBER-10(NEGATIVE BECAUSE IT IS A RESERVED SYSTEM  %
    %           EVENT), WILL BE SET AFTER A POWER FAIL/RESTART.            %
ENDPROC;
```

```
TITLE
      SMT OPERATING SYSTEM
      ( WITH NEGATIVE SYSTEM TASK NUMBERS )
      MACHINE DEPENDENT SYSTEM ROUTINES (READ ONLY)
      **** MODULE SMTB1 ****
      SMT 1(18) 13-04-1981;

OPTION (1);

LET OVHD       =  30;                  % STACK OVERHEAD FOR SVCS ETC    %
LET RN         =   STKUSG+2;           %                               %
LET MASK       =   OCT 340;            %                               %
LET UNMSK      =   0;                  %                               %
LET HALT       =   0;                  %                               %

LET NTASKS     =  22;                  %                               %
LET NSTSK      =   6;                  %                               %
LET NTDV16     =   2;                  %                               %
LET NEVENTS    =  32;                  %                               %
LET NSEV       =  16;                  %                               %
LET NFAC       =  32;                  %                               %
LET NSFAC      =  16;                  %                               %
LET NEVQS      =  15;                  %                               %
LET INVEVS     = -16;                  %                               %
LET EVQLEN     =  16;                  %                               %

LET WTG        =   2;                  %                               %
LET WTSEC      =   3;                  %                               %
LET STOPP      =   OCT 100;            %                               %

LET QFULLERROR=  39;                   % ERROR NO. FOR EVENT QUEUE FULL%
LET CLKTASKNO =  -2;                   %                               %
LET SETASKNO  =  -3;                   %                               %
LET ERPTNO     =  -1;                  % ERROR PRINT TASK NO            %
LET PWFEV      = -10;                  % POWER FAIL EVENT               %
LET NSTCON     =  16;                  % NO. OF STACK VALUES PRINTED    %
                                       % ON ERROR                       %

MODE DELREC ( INT TIMUP, TASK, REF DELREC NXT );

MODE TASKLINK ( REF ARRAY STACK TKSTACK,
                REF ARRAY PROC() TKPROC,
                REF ARRAY BYTE TKPRIO,
                              TKSTAT,
                PROC () USERINITS,
                        USERPF
              );


   % ++++++++++ MARKS PROCESSOR STATUS WORD ACCESS                      %

%NOTE THAT THE CODE FOR SETTING THE PSW ON THE LSI IS SHORTER THAN      %
%ON 11/34 ETC AND ADJUST LENGTHS OF CODE SECTIONS ACCORDINGLY.          %
%THE CONSTRUCTIONS RECOMMENDED ALLOW THE CODE TO BE HELD IN ROM.        %
%(MTPS #340 WILL ONLY WORK IN RAM - 1976-77 DEC MICROCOMPUTER HANDBOOK %


EXT STACK SYSTACK,FBSTK,HSTK;

EXT DATA TIMEDATA;
   INT NOW,SECSNOW,MINSNOW,TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;
```

```
EXT DATA CLKDATA;
   INT TICK;                              % COUNT ALLOWS CLOCK TO CATCH   %
ENDDATA;                                  % UP IF TASKLOCK ON             %

EXT DATA TASKDATA;
   INT CURTASK,CURTEX,TASKLOCK,NXTCUR,EVIN,EVOUT;
   DELREC ADEL;
   REF DELREC FRPTR;
   BYTE HIPRI,LOPRI;
   ARRAY (NTASKS) INT TIMEOUT;
   ARRAY (NTASKS) DELREC DEL;
   ARRAY (NTASKS) STACK  CELL;
   ARRAY (NTASKS) BYTE EVFAC,STAT,PRIO,WTCHN;
   ARRAY (NTDV16,NEVENTS) INT EVBITS;
   ARRAY (NTASKS) REF BYTE STATADS;
   ARRAY (EVQLEN) INT EVQ;
   ARRAY (NFAC) BYTE FACILITY,FACTOTSK;
   ARRAY (NEVENTS) BYTE EVTOTSK;
ENDDATA;

EXT DATA REGDATA;
   ARRAY(9)INT REGS;
   ARRAY(NSTCON)INT STKVALS;
   INT ECT,ERNO,TASKNO,LINENO;
ENDDATA;

EXT DATA TRAPDATA;
   LABEL TASKEXIT;
   REF TASKLINK T;
   INT UR1,ERNUM,UPS;
ENDDATA;

EXT DATA PWFDATA;
   INT PWFLAG;
ENDDATA;


EXT PROC () UNLOCK;
EXT PROC () CLEANUP;
EXT PROC (INT) DFERP,STOP;
EXT PROC () BYTE DEFIN;
EXT PROC (BYTE) DEFOUT;
EXT PROC () RRNUL;

SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT ENDDATA;
SVC DATA RRSED; BYTE TERMCH,IOFLAG; ENDDATA;
SVC DATA STKUSG; INT USAGE,LINE ENDDATA;


%%%%%%%%%  RZZ  %%%%%%%%%

   % THIS DATA BRICK IS INCLUDED TO PRESERVE THE STANDARD 'MTS'          %
   % INTERFACE TO THE CONTROL ROUTINES. IT COULD BE REMOVED IF           %
   % 'SMT' WRE MOUNTED ON ANOTHER M/C                                    %

ENT DATA RZZ;
   PROC (INT) SYSERRORPROC:=RRGEL;
   INT ZZ1:=ZZ2:=0;
   REF INT ZZ3:=ZZ1;    % LOCATION R00+6 IS USED AS THE ENTRY ADDRESS    %
                        % FOR SVC PROCEDURE CALLS (MTS PROC 'CHANGE)     %
```

```
                        % SUCH CALLS IN SMT WILL THEREFORE HALT AT      %
                        % ADDRESS R00+2                                 %
ENDDATA;

OPTION (2) CM,SL;

   % ALL THE PSEUDO PROCEDURES WHICH FOLLOW, OVERWRITE THE NORMAL       %
   % RTL/2 PROCEDURE ENTRY CODE. THIS IS ASSUMED TO BE 6 BYTES          %
   % LONG (HENCE OPTION 'SL' ABOVE) AND THE CODE SECTION LENGTH         %
   % IS REDUCED ACCORDINGLY                                             %


PROC INTPTS();

   %%%%%%%%%  DEFINE SVC DATA BRICKS %%%%%%%%%

   CODE 0,0;
   RRSIO==6
   RRERR==10.
   RRSED==18.
   STKUSG==20.
   &CR N U,RRSIO=6,PQBPBQ0Z
   &CR N U,RRERR=10.,LIPIQ0Z
   &CR N U,RRSED=18.,BT2Z
   &CR N U,STKUSG=20.,IT2Z

   ; THE FOLLOWING 'CONTROL ROUTINE' NAMES ARE
   ; ASSUMED BY THE STANDARD MTS CONTROL ROUTINE MODULES
   R00==RZZ
   &CR R00=RZZ
   &CR R23=RZZ+2
   &CR R24=RZZ+2
   *RTL;

%%%%%%%%% H - TASK NOTES %%%%%%%%%


% REFER TO MANUAL FOR FULL SYNTAX.                                      %


%   SMT  (ALL VERSIONS)   RTL/2 IN H - TASKS                            %
%   ========================================                            %

% CERTAIN RTL/2 CONSTRUCTIONS CAN CAUSE THE COMPILER TO USE WORKSPACE   %
% ABOVE THE H - STACK.                                                  %
% FOR EXAMPLE, EVALUTING A COMPLEX EXPRESSION FOR LOOP COUNTING         %
% LOCAL VARIABLES, OR BLOCK .... DECLARATION .... ENDBLOCK.             %
% ON INTERRUPT THE VALUE OF R5 IS ONLY THE DUMMY HALF LINK CELL ON THE  %
% TOP OF THE STACK, AND ANY DISPLACEMENT FROM THIS CORRUPTS WHATEVER    %
% IS LINKED ABOVE IT.                                                   %
%                                                                       %
% THE SECURE SOLUTION IS TO CREATE A PROCEDURE AND CALL IT THUS:-       %

% PTORTL();                                                             %
% MYHPROC();                                                           %

% THIS CALLS R01 AND CREATES A NEW LINK CELL AND SPACE ON THE           %
% H - STACK FOR LOCAL VARIABLES AND WORKSPACES.                         %
% RETFIN();                                                            %


% ALTERNATIVELY, IF THE DESIRED SPACE IS DETERMINED AS BELOW THEN       %
```

```
% SPACE MAY BE CREATED ABOVE THE H - STACK.                              %
% FOR EXAMPLE:-                                                          %
% ENT STACK HSTK 100;                                                    %
% DATA HWORKSPACE;                                                       %
% INT R7,FDUMP4,FDUMP6,FDUMP10;                                          %
% ENDDATA;                                                               %
% ENT STACK .......                                                      %

% THIS AMOUNT ( 4 WORDS ) SHOULD COVER MOST CASES.                       %

% IF THE PAL OUTPUT BY THE COMPILER IS INSPECTED AND THE PROCEDURE       %
% ENTRY CODE ( WHICH IS OVERWRITTEN ) IS                                 %
%       JSR   2,R01                                                      %
%       2,177774                                                         %
% THEN NO WORKSPACE IS USED AND EXECUTING THE CODE IS SAFE WITHOUT       %
% CREATING ANY WORKSPACE.                                                %


% IF THE CODE BETWEEN PTORTL AND ENDPROC IS INSPECTED,                   %
% THE HIGHEST DISPLACEMENT ON REGISTER 5  E.G. 6(5) INDICATES            %
% THE WORKSPACE ACTUALLY NEEDED AND THIS NUMBER DIVIDED BY TWO           %
% GIVES THE NUMBER OF WORDS OF WORKSPACE NEEDED, THAT IS 3 FOR THIS      %
% EXAMPLE.                                                               %
%                                                                        %



   %%%%%%%%%  VEC 4   INTERRUPT   %%%%%%%%%

   CODE 26,0;                          ; +++++++++ 26 FOR LSI

      .=*INTPTS
VEC4:
     MOV #30.,*ERNUM/TRAPDATA
VEC4.4:                                ; COMMON UNRECOV ERROR CALL
      MOV %1,*UR1/TRAPDATA             ; SAVE USERS %1
      MOV (6)+,%1                      ; USERS LINK IN %1 (JSR 1,RRGEL)
      MOV (6),*UPS/TRAPDATA            ; USERS PS
      MOV *ERNUM/TRAPDATA,(6)          ; ERROR NUMBER (PARAM OF RRGEL
      MOV *UR1/TRAPDATA,-(6)
      MTPS *UPS/TRAPDATA    ; RESTORE USERS PS
                                       ; MTPS *UPS/TRAPDATA ON LSI
      JMP @#*RRGEL                     ; SIMULATES JSR 1, RRGEL FROM USER
   *RTL;


   %%%%%%%%%  VEC 10   INTERRUPT   %%%%%%%%%

   CODE 8,0;
VEC10:
      MOV #33.,*ERNUM/TRAPDATA
      BR VEC4.4
   *RTL;


   %%%%%%%%%  VEC 20   INTERRUPT   %%%%%%%%%

   CODE 8,0;
VEC20:
      MOV #34.,*ERNUM/TRAPDATA
      BR VEC4.4
   *RTL;
```

```
   %%%%%%%%%  VEC 30  INTERRUPT  %%%%%%%%%

   CODE 8,0;
VEC30:
      MOV #35.,*ERNUM/TRAPDATA
      BR VEC4.4
   *RTL;

   %%%%%%%%%  POWER FAIL/RESTART INTERRUPT  %%%%%%%%%

   CODE 18,0;
VEC24:                    ; POWER FAIL ENTRY
PW:
      JSR %1,*PTORTL      ; SAVE CURRENT MACHINE STATE
      MOV #PW2,24         ; POINT VECTOR AT POWER UP CODE
      HALT
PW2:                      ; POWER RESTORE ENTRY
      MOV #*BEGIN+4,%1    ; SKIP OVER COLD START ENTRY POINT  **RSX**
      JMP (1)             ; %1 NON ZERO IS ALSO USE AS A POWER FAIL
                          ; FLAG BY THE STARTUP CODE
   *RTL;

   %%%%%%%%%  LINE TRACE INTERRUPT (BPT)  %%%%%%%%%

   CODE 12,0;
VEC14:
      MOV   @0(6),*LINE/STKUSG(0)   ; LINE NUMBER
      ADD   #2,(6)                  ; MOVE USER LINK PAST IT
      RTI
   *RTL;
ENDPROC;


ENT PROC CLOCKINT();
   % CLOCK INTERRUPT SERVICING CODE                                    %
   CODE 14,0;
         .=CLOCKINT                    ; OVERWRITES ENTRY CODE
         INC   *NOW/TIMEDATA           ; TICK UP NOW
         INC   *TICK/CLKDATA           ;
         CLRB  *STAT/TASKDATA+*NSTSK+*CLKTASKNO ;
         TST   *TASKLOCK/TASKDATA      ;
         BEQ   CKHT                    ;
         RTI                           ; RETURN IF LOCKED
    CKHT:
   *RTL;

   PTORTL();                          % SWITCH TO H-TASK ENVIRONMENT  %
      CURTEX:=CLKTASKNO+NSTSK;
   RETFIN();
ENDPROC;

%%%%%%%%%  RRGEL  %%%%%%%%%


ENT PROC RRGEL(INT N);

   % HAND CODED UNRECOVERABLE ERROR PROC                               %

   CODE 152,4;                   ; ++++++++ 152 FOR LSI
```

```
   .=*RRGEL
   MFPS -(6)              ; USERS PS  +++++++ MFPS -(6)
   MOV %1,-(6)                      ; USERS LINK ADDR
   MOV 6(6),*ERN/RRERR(0)           ; SET UNREC ERROR NO
   MTPS *HIPRI/TASKDATA              ; LOCKOUT +++++++ MTPS *HIPRI/TASKDATA
   CLRB *STAT/TASKDATA+*ERPTNO+*NSTSK ; ERROR PRINT TASK TO GO
   INC *ECT/REGDATA
   CMP *ECT/REGDATA,#1
   BGT RRGL.1                       ; AN ERROR ALREADY RECORDED
   MOV *LINE/STKUSG(0),*LINENO/REGDATA
   MOV *CURTASK/TASKDATA,*TASKNO/REGDATA
   CMP %0,#*HSTK
   BNE RRGL.0
   CLR *TASKNO/REGDATA
RRGL.0:
   MOV 6(6),*ERNO/REGDATA           ; ERROR NUMBER FROM STACK
   MOV #*REGS/REGDATA+20.,%1        ; TRANSFER REGISTERS
   MOV 2(6),-(1)
   MOV (6),-(1)                     ; USERS LINK (%7)
   MOV %6,-(1)
   ADD #8.,(1)                      ; USERS %6 BEFORE ENTRY TO RRGEL
   MOV %5,-(1)
   MOV %4,-(1)
   MOV %3,-(1)
   MOV %2,-(1)
   MOV 4(6),-(1)                    ; SAVED USERS %1
   MOV %0,-(1)
                                    ; USER'S STACK BEFORE ENTRY
   MOV 14(1),%2
   MOV #*STKVALS/REGDATA+2,%1
RRGL.3:
   MOV (2)+,(1)+                    ; TRANSFER STACK VALUES TO STKVALS
   CMP %1,#*STKVALS/REGDATA+*NSTCON+*NSTCON
   BLE RRGL.3
RRGL.1:
   ADD #8.,%6                       ; STACK NOW EMPTY
   CMP %0,#*HSTK
   BEQ HTASKERR                     ; UNRECOV ERROR IN H-TASK
   MOV *ERL/RRERR+2(0),-(6)         ; FINAL %5
   MOV #RRGL.2,-(6)                 ; GO TO RRGL.1 IF UNWIND OK
   TRAP 25.                         ; UNWIND STACK
   MOV #3,*ERNO/REGDATA             ; STACK UNWIND FAILURE
   MOV *TASKEXIT/TRAPDATA,*ERL/RRERR(0)   ; DEFAULT %7
   MOV %0,%5                        ; RESET STACK POINTER
   ADD (0),%5                       ; SO THAT CLEANUP WILL WORK
   MOV %5,%6
RRGL.2:
   CLR *TASKLOCK/TASKDATA           ; REMOVE TASKLOCK
   MTPS *LOPRI/TASKDATA             ; ++++++++ MTPS *LOPRI/TASKDATA
   JMP @*ERL/RRERR(0)               ; GOTO ACTUAL ERROR LABEL
HTASKERR:
   JSR %1,*RETFIN                   ; H-TASK UNRECOV ERROR EXIT
   *RTL;
ENDPROC;


%%%%%%%%%  INTERRUPT LOCKOUT - HLOCK AND HUNLOCK  %%%%%%%%%


ENT PROC HLOCK();
   CODE 0,0;                        ; ++++++++ 0 FOR LSI
      .=*HLOCK
```

0-5-6

```
      MTPS *HIPRI/TASKDATA       ;++++++++ MTPS *HIPRI/TASKDATA
      RTS   %1
   *RTL;
ENDPROC;

ENT PROC HUNLOCK();
   CODE 0,0;                      ; ++++++++ 0 FOR LSI
      .=*HUNLOCK
      MTPS *LOPRI/TASKDATA      ;++++++++ MTPS *LOPRI/TASKDATA
      RTS   %1
   *RTL;
ENDPROC;

%%%%%%%%%  INITIAL STARTUP ROUTINES  %%%%%%%%%

PROC BEGIN();
   % M/C SPECIFIC STARTUP CODE FOR PDP-11                                  %
   % SET UP INITIAL ENTRY (FROM ZERO) AND ALL                              %
   % INTERRUPT VECTORS STATICALLY                                          %
   % INITIALISE STARTUP TASK STACK AND THEN CONVERT                        %
   % TO A PROPER TASK BY CALLING RRMAG                                     %

   CODE 60,0;                 ;++++++++ 60 FOR LSI  **RSX**
SMT...==*BEGIN

   .ASECT
   .=0
   JMP @#*BEGIN               ; ENTRY FROM ZERO
   .WORD VEC4,*MASK           ; VEC4 TRAP
   .WORD VEC10,*MASK          ; VEC10 TRAP
   .WORD VEC14,*MASK          ; VEC14 TRAP (LINE TRACE)
   .WORD VEC20,*MASK          ; VEC20 TRAP
   .WORD VEC24,*MASK          ; VEC24 TRAP (POWER FAIL)
   .WORD VEC30,*MASK          ; VEC30 TRAP
                              ; VEC34 TRAP IS INITIALISED BY CONTROL RTNS
   .=40                       ;
   .WORD 0                    ; THIS LOCATION USED FOR TKLINK
                              ; INITIALISED BY SMTU1

; INTERRUPT VECTORS FOR DEVICES - THESE WILL BE OVERWRITTEN AT LOAD
; TIME FOR THOSE DEVICES FOR WHICH THERE ARE INTERRUPT HANDLING ROUTINES

   .=60
         .+2,*HALT            ; TTY KBD
         .+2,*HALT            ; TTY PRINTER
         .+2,*HALT            ; HS PT RDR
         .+2,*HALT            ; HS PT PUNCH
         *CLOCKINT,*MASK      ; CLOCK - ASSUMES OPTION 'SL'
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
         .+2,*HALT
```

0-5-7

0-5-8

```
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
           .+2,*HALT
     .PSECT SMTB1X

     .=*BEGIN
        RESET                      ;    **RSX**
        CLR %1                     ; DISTINGUISH COLD START FROM
                                   ; POWER FAIL RESTART
        MOV 40,*T/TRAPDATA         ; PICK UP ADDRESS OF USER TKLINK DATA
        BNE *L0
        HALT                       ; USER MODULE NOT LOADED
  *L0*:
        MOVB #*MASK,*HIPRI/TASKDATA
        CLRB *LOPRI/TASKDATA
        MTPS *HIPRI/TASKDATA        ; ++++++++ MTPS *HIPRI/TASKDATA
        MOV #*HSTK,%0              ; STACK ADDR
  *L1*:
        MOV %0,%5
        ADD (0),%5                 ; END OF STACK ADDR
        MOV %5,%6
        MOV %6,(6)                 ; INITIALISE LAST (HALF) LINK CELL
        MOV %0,2(0)                ;) STACK LENGTH CHECK
        ADD #*OVHD+*RN,2(0)        ;) WORD SKLO
        CMP #*HSTK,%0
        BNE *L2
        MOV #PW,24                 ; SET POWER FAIL INTERRUPT VECTOR
        TST %1                     ; POWER FAIL RESTART ?
        BEQ *NPF                   ; BRANCH IF NORMAL COLD START
     *RTL;

     % POWER FAIL RESTART                                                     %
```

```
   PWFLAG := PWFLAG + 1;
   QEV(PWFEV);
   T.USERPF();                          % CALL USER DEFINED POWER FAIL  %
                                        %RESTART PROC                  %

NPF:
   CODE 6,0;
      MOV   #*SYSTACK,%0      ; INITIALISE SYSTEM STACK
      BR    *L1
   *RTL;
L2:
   RRMAG(SETFMQ);                       % SETFMQ WILL INITIALISE ALL    %
                                        %S-TASK STACKS                 %
ENDPROC;


ENT PROC TYPCHANGE();
   % NULL PROCEDURE USED TO PROVIDE A COMMON FRAMEWORK FOR THE         %
   % RTL TYPE CHANGE PROCEDURES SUCH AS                               %
   %       ENT PROC ITORI(INT I)REF INT;                              %

   CODE 0,0;
      .=*TYPCHANGE
      RTS   %1

      ITORI==*TYPCHANGE
      ITORF==*TYPCHANGE
      ITORR==*TYPCHANGE
      ITORB==*TYPCHANGE
      RBTOI==*TYPCHANGE
      &CR N P,ITORI,IQEIZ        ; INT TO REF INT
      &CR E ITORI
      &CR N P,ITORF,IQEFZ        ; INT TO REF FRAC
      &CR E ITORF
      &CR N P,ITORR,IQERZ        ; INT TO REF REAL
      &CR E ITORR
      &CR N P,ITORB,IQEBZ        ;  INT TO REF BYTE
      &CR E ITORB
      &CR N P,RBTOI,EBQIZ        ;  REF BYTE TO INT
      &CR E RBTOI
   *RTL;
ENDPROC;


ENT PROC STKINIT(STACK STK,PROC () TKP);
   CODE 46,0;
      MOV   *STK(5),%1
      MOV   %1,%2          ; ADDRESS OF STACK
      ADD   (1),%2         ; ADDRESS OF LAST WORD - USERS %5
      MOV   %2,(2)         ; LAST (HALF) LINK CELL
      MOV   %2,%4
      MOV   *TKP(5),-(2); USER BASE PROC (FOR RRMAG)
      TST   -(2)
      MOV   #*UNMSK,-(2); PS INITIAL VALUE
      MOV   #*RRMAG,-(2); %7 INITIAL VALUE
      SUB   #8.,%2         ; SKIP STORAGE OF %1-%4
      MOV   %4,-(2)        ; %5 INITIAL VALUE
                           ; N.B. WHEN USING HARDWARE FLOATING POINT UNIT
                           ;      EXTRA SPACE WILL BE REQUIRED TO STORE
                           ;      THE FLOATING POINT ACCUMULATORS
      MOV   %2,4(1)        ; SX6
      MOV   %1,2(1)        ; STKLO-USED FOR STACK LENGTH CHECKS
```

```
      ADD   #*OVHD+*RN,2(1)
   *RTL;
ENDPROC;

ENT PROC SVDINIT();
   % SVC DATA INITIALISATION PROCEDURE                                    %

   % SET UP TASK DEFAULT ERROR LABEL                                      %
   % X7=TASKXT IN RRMAG                                                   %
   % X5=STACK END                                                         %
   CODE 20,0;
      MOV   *TASKEXIT/TRAPDATA,*ERL/RRERR(0)
      MOV   %0,*ERL/RRERR+2(0)
      ADD   (0),*ERL/RRERR+2(0)
      MOV   #177777,*LINE/STKUSG(0)
   *RTL;
   ERP:=DFERP;
   IN:=DEFIN;  OUT:=DEFOUT;
ENDPROC;

ENT PROC PTORTL();

   CODE 18,0;
R24==*PTORTL
   .=*PTORTL
   ; SAVE USERS REGISTERS AND SWITCH TO HSTK
   ; 'JSR %1,PTORTL'   HAS ALREADY PUT USERS %1 ON THE STACK
   MOV %2,-(6)
   MOV %3,-(6)
   MOV %4,-(6)
   MOV %5,-(6)
   MOV %6,4(0)                    ; %6 INTO SX6 ON STACK
   MOV #*HSTK,%0                  ; HARDWARE TASK STACK
   MOV %0,%5
   ADD (0),%5                     ; POINT TO LAST (HALF) LINK CELL
   MOV %5,%6
   JMP (1)
   *RTL;
ENDPROC;

ENT PROC RETFIN();

   CODE 30,0;
R23==*RETFIN
   .=*RETFIN
   ; TERMINATE H-TASK AND RETURN TO S-TASK DENOTED BY CURTEX
   MOV *CURTEX/TASKDATA,%3
   MOV %3,*CURTASK/TASKDATA
   SUB #*NSTSK,*CURTASK/TASKDATA
   ASL %3
   MOV *CELL/TASKDATA(3),%0       ; NEW STACK ADDR:=CELL(CURTEX)
   MOV 4(0),%6                    ; SX6
   MOV (6)+,%5
   MOV (6)+,%4
   MOV (6)+,%3
   MOV (6)+,%2
   MOV (6)+,%1
   RTI                            ; EXIT TO USER TASK
   *RTL;
ENDPROC;
```

```
%%%%%%%%% CHANGE %%%%%%%%%

ENT PROC CHANGE();
   CODE 44,0;                        ;++++++++ 44 FOR LSI

   .=*CHANGE
   MFPS   (6)              ; PSW TO STACK ++++++++ MFPS (6)
   MTPS *HIPRI/TASKDATA     ; ++++++++ MTPS *HIPRI/TASKDATA
   MOV   %1,-(6)             ; LINK - SIMULATE HINTERRUPT
   SUB   #10,%6              ; SET UP CELL WITH ONLY
   MOV   %5,-(6)             ; MEANINGFUL REGISTERS TO SAVE
   MOV   %6,4(0)             ; TIME OVER JSR 1 PTORTL
   CLR   *TASKLOCK/TASKDATA
   CMP   *EVIN/TASKDATA,*EVOUT/TASKDATA
   BNE   *RETEV+10           ;UNPACK EVENT Q IF REQUIRED
   MOV   #*STATADS/TASKDATA+2,%4; FIRST STATUS ADDRESS - HIGH PRIO
TSKTST:
   TSTB  @(4)+               ; TEST CONTENTS OF STAT(STADS(I))
   BNE   TSKTST              ; FOR ACTIVE TASK
   MOV   -(4),%3             ; ADDRESS OF CURTEX(STAT)
   SUB   #*STAT/TASKDATA,%3  ; CURTEX
   MOV   %3,*CURTEX/TASKDATA
   BR    *RETFIN+4           ; QUICK JUMP TO RETFIN
   *RTL;
ENDPROC;


OPTION (1) ;

% THE FOLLOWING ROUTINES - QEV,QEVRRET,RETEV,CLOCK,SETFMQ AND RRMAG -  %
% COULD RESIDE IN MODULE SMTB2. THEY ARE INCLUDED HERE BECAUSE         %
% THEY COULD BE HAND CODED IN SOME SYSTEMS                             %

ENT PROC RETEV();
   % USED TO RETURN FROM INTERRUPT                                     %
   % IF AN EVENT HAS BEEN SET                                          %
   IF TASKLOCK=0 THEN
      CURTEX:= SETASKNO+NSTSK;          % ENSURE EVENT QUEUE IS SERVICED%
      TASKLOCK:=TASKLOCK-1;
   END;
   RETFIN();                            % RETURN                       %
ENDPROC;


ENT PROC QEV(INT EVENT);
   % USED BY INTERRUPT SERVICING CODE TO                               %
   % PLACE AN EVENT IN THE QUEUE                                       %
   INT EVTST:=(EVIN+1) LAND NEVQS;
   IF EVTST=EVOUT THEN
      RRGEL(QFULLERROR);
   END;
   EVQ(EVTST+1):= EVENT;
   EVIN:=EVTST;
ENDPROC;

ENT PROC QEVRRET(INT EVENT);
   % PLACE AN EVENT IN THE QUEUE                                       %
   % AND ALSO RETURN FROM INTERRUPT                                    %
   QEV(EVENT);
   RETEV();
ENDPROC;
```

0-5-12

```
ENT PROC CLOCK();
   % BASE PROCEDURE FOR CLOCK INTERRUPT TASK                          %
CLK:
   WHILE TICK > 0 DO
      TICK := TICK - 1;
   IF TCOUNT>=49 THEN
      TCOUNT:=0;  SECSNOW:=SECSNOW+1;
      IF SECS>=59 THEN
         SECS:=0; MINSNOW:=MINSNOW+1;
         IF MINS>=59 THEN
            MINS:=0;
            IF HOURS>=23 THEN
               HOURS:=0;              % MIDNIGHT OWLS AND BROOMSTICKS %

   % EVEN MONTHS ARE THIRTY DAY                                       %
   % UP TO AUGUST ANYWAY                                             %
   % POST JULY WE JUST ADD 1                                         %
   % AND FINISH AS WE HAVE BEGUN                                     %
   % THIRTY NEEDS BITS FIVE TO TWO                                   %
   % SO LOR THEM IN - BIT ONE DROPS THROUGH                          %
   % A SPECIAL TEST FOR FEBRUARY                                     %
   % I'M SAD TO SAY IS NECESSARY                                     %
   % BUT ALL THE REST, NEW YEAR TO YULE                              %
   % ARE GIVEN BY THIS SIMPLE RULE.                                  %
                                    %A.J.C. 1978                     %

            IF DAYS>=IF MONTHS=2 THEN  % FILLDYKE                    %
            IF YEARS LAND 3 = 0 THEN   % LEAPYEAR                    %
                              29
                           ELSE 28
                           END
                        ELSE 30 LOR IF MONTHS > 7 THEN MONTHS+1
                                    ELSE MONTHS END
                        END
                                    THEN
                  DAYS:=0;
                  IF MONTHS>=12 THEN  % HOGMANAY                     %
                     MONTHS:=0;  YEARS:=YEARS+1;
                  END;
                  MONTHS:=MONTHS+1;
               END;
               DAYS:=DAYS+1;
          ELSE
               HOURS:=HOURS+1;
          END;
         ELSE
            MINS:=MINS+1;
         END;
      ELSE
         SECS:=SECS+1;
      END;
   ELSE
      TCOUNT:=TCOUNT+1;
   END;
REP;
   % COUNT DOWN DELAYS AND TIMEOUTS                                   %

   TASKLOCK:=1;
TRIAG:
   BLOCK
      REF DELREC PTR:=ADEL.NXT;
      INT I, S, TIMTST;
```

```
      IF PTR :=: ADEL THEN GOTO STP; END;
      TIMTST := PTR.TIMUP - NOW;
   IF TIMTST > 0 THEN  GOTO STP; END;  % TIMEUP                          %
      I := PTR.TASK;
      ADEL.NXT := PTR.NXT;
      PTR.NXT := FRPTR;
      FRPTR := PTR;
       S := STAT(I)LAND WTSEC;
   IF S # 0                            % TIMEOUT ONLY COUNTED IN WAIT  %
                                       % PERIOD                        %
   THEN IF TIMEOUT(I) > 0
      THEN TIMEOUT(I) := 0;            % TIMEOUT HAS RUNOUT, SO CLEAR  %
                                       % TASK FROM WTCHN               %
   BLOCK
        REF BYTE TKPTR := IF S = WTG THEN EVTOTSK(EVFAC(I))
                                     ELSE FACTOTSK(EVFAC(I))
                                        END;
            WHILE TKPTR # I            % TASK MUST BE IN WTCHN         %
            DO TKPTR := WTCHN(TKPTR)   % IF TO > 0 AND STATUS WAITING  %
            REP;                       % FOR FACILITY OR EVENT         %
            VAL TKPTR := WTCHN(I);
            WTCHN(I) := 0;             % REMOVES TASK FROM WTCHN       %
            EVFAC(I) := 0;
      ENDBLOCK;
   END;
END;
      STAT(I):=STAT(I) LAND STOPP;
      GOTO TRIAG;
   ENDBLOCK;
STP:
   STOP(CLKTASKNO);                    % NB STOP CALLS CHANGE          %
   GOTO CLK;
ENDPROC;

ENT PROC SETFMQ();
   LABEL RERL;
   ADEL.NXT := ADEL;
   FRPTR := DEL(1);

   FOR I:=1 TO NTASKS DO
      TIMEOUT(I):=-1;
      DEL(I).NXT := IF I = NTASKS
                    THEN ADEL
                    ELSE DEL(I+1)
                    END;
      CELL(I):=FBSTK;
      PRIO(I):=WTCHN(I):=EVFAC(I):=0;
      STAT(I):=STOPP;
      STATADS(I):=STAT(1);
   REP;
   FOR I := 1 TO NTDV16 DO
      FOR J := 1 TO NEVENTS DO
         EVTOTSK(J) := 0;  EVBITS (I,J) := 0;
      REP;
   REP;
   FOR I:=1 TO NFAC DO FACILITY(I):=0; FACTOTSK(I) := 0 REP;
   CURTASK:=SETASKNO;
   CURTEX:=SETASKNO+NSTSK;
   TASKLOCK:=EVIN:=EVOUT:=ECT:=0;

   FOR I:=1 TO NTASKS DO
      PROC()TKP := T.TKPROC(I);
```

```
      IF TKP#RRNUL THEN
         PRIO(I):=T.TKPRIO(I);
         STAT(I):=T.TKSTAT(I);
         CELL(I):=T.TKSTACK(I);
         IF I#SETASKNO+NSTSK THEN      % AVOID REINITIALISING          %
                                       % OWN STACK                    %
            STKINIT(CELL(I),TKP);
         END;
      END;
   REP;

   % INITIALISE 'STATADS' ARRAY USED BY CHANGE                         %

   BLOCK
      INT J:=0,MAXP:=256,P;
L:
      P:=-1;
      FOR I:=1 TO NTASKS DO
         INT P1:=PRIO(I);
         IF P1>P AND P1<MAXP THEN P:=P1 END;
      REP;
      IF P>=0 THEN
         FOR I:=1 TO NTASKS DO
            IF PRIO(I)=P THEN
               J:=J+1;
               STATADS(J):=STAT(I);
            END;
         REP;
         MAXP:=P;
         GOTO L;
      END;
   ENDBLOCK;

   RERL:=ERL;
   ERL:=UERL;
   T.USERINITS();                    % PERFORM USER INITIALISATIONS  %
UERL:
   ERL:=RERL;

   % END OF STARTUP CODE                                               %

   HUNLOCK();
   CHANGE();                         % SELECT A NEW S-TASK           %

   % STARTUP 'TASK' NOW REVERTS TO STANDBY LOOP                        %
   % WHICH SERVICES THE EVENT QUEUE                                   %

   ERL:=EMTQ;
EMTQ:
   UNLOCK();
   GOTO EMTQ;
ENDPROC;


ENT PROC RRMAG(PROC () P);
   % BASE PROCEDURE FOR ALL TASKS                                      %

   TASKEXIT:=TASKXT;
   SVDINIT();
   ERN:=0;  IOFLAG:=0;
```

```
   P();                                  % RUN USER PROVIDED BASE          %
                                         % PROCEDURE                       %
   % NO TASK SHOULD EVER EXIT                                             %

   SVDINIT();                            % RESET ERL TO JOBXT               %
   RRGEL(10);

TASKXT:                                  % TASK DEFAULT UNRECOVERABLE      %
                                         % ERROR LABEL                     %

   CLEANUP();
   STOP(CURTASK);
   GOTO TASKXT;
ENDPROC;
```

```
TITLE
      SMT OPERATING SYSTEM
      MACHINE INDEPENDENT SYSTEM ROUTINES (READ ONLY)
      DELAY, TIMEOUT AND WAIT, FOR EVENTS AND
      FACILITIES IN ORDERED LISTS FOR SPEED
      **** MODULE SMTB2 ****
      SMT 2(18) 13-04-1981;

OPTION (1) ;

LET NL          =  10;              %                                  %
LET SP          =  32;              %                                  %
LET TAB         =   9;              %                                  %
LET EOM         =   3;              %                                  %
LET ENQ         =   5;              %                                  %
LET BELL        =   7;              %                                  %
LET CTLAEV      = -15;              % CTLA TASK EVENT                   %
LET NEVQS       =  15;              %                                  %
LET NFAC        =  32;              %                                  %
LET NSFAC       =  16;              %                                  %
LET NTASKS      =  22;              %                                  %
LET NSTSK       =   6;              %                                  %
LET NTDV16      =   2;              %                                  %
LET NEVENTS     =  32;              %                                  %
LET NSEV        =  16;              %                                  %
LET FACEV       = -11;              % EVENT WAKES UP SECURED TASKS      %
LET STOPP       =  OCT 100;         %                                  %
LET SUSP        =  OCT 200;         %                                  %
LET WTG         =   2;              %                                  %
LET WSEC        =   1;              %                                  %
LET NOTSTOP     =  OCT 277;         %                                  %
LET EVQLEN      =  16;              %                                  %
LET NSTCON      =  16;              % NO. OF STACK VALUES PRINTED       %
                                    % ON ERROR                         %
LET ERPTNO      =  -1;              % ERROR PRINT TASK NO               %

MODE DELREC ( INT TIMUP, TASK, REF DELREC NXT );


EXT PROC ()HLOCK,HUNLOCK;
EXT PROC (INT)  RRGEL;
EXT PROC () CHANGE;
EXT PROC (INT) REF INT ITORI;
EXT PROC (INT) REF FRAC ITORF;
EXT PROC (INT) REF REAL ITORR;
EXT PROC (INT) REF BYTE ITORB;

SVC DATA RRERR; LABEL ERL; INT ERN; PROC (INT) ERP ENDDATA;
SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT ENDDATA;
SVC DATA RRSED; BYTE TERMCH,IOFLAG ENDDATA;

EXT DATA TASKDATA;
   INT CURTASK,CURTEX,TASKLOCK,NXTCUR,EVIN,EVOUT;
   DELREC ADEL;
   REF DELREC FRPTR;
   BYTE HIPRI,LOPRI;
   ARRAY (NTASKS) INT  TIMEOUT;
   ARRAY (NTASKS) DELREC DEL;
   ARRAY (NTASKS) STACK  CELL;
   ARRAY (NTASKS) BYTE EVFAC,STAT,PRIO,WTCHN;
   ARRAY (NTDV16,NEVENTS) INT EVBITS;
```

```
   ARRAY (NTASKS) REF BYTE STATADS;
   ARRAY (EVQLEN) INT EVQ;
   ARRAY (NFAC) BYTE FACILITY,FACTOTSK;
   ARRAY (NEVENTS) BYTE EVTOTSK;
ENDDATA;

EXT DATA TIMEDATA;
   INT NOW,SECSNOW,MINSNOW,TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

EXT DATA REGDATA;
   ARRAY(9)INT REGS;
   ARRAY(NSTCON)INT STKVALS;
   INT ECT,ERNO,TASKNO,LINENO;
ENDDATA;

EXT DATA SYSIODATA;
   PROC()BYTE CTLAIN;
   PROC(BYTE) CTLAOUT,ERROUT;
ENDDATA;

ENT DATA PATTERNS;                          % BIT MASK PATTERN DATA          %
                                            % (DO NOT ALTER***)              %
   ARRAY (16) INT BITS:=(1,2,4,8,16,32,64,128,
                         256,512,1024,2048,4096,8192,16384,HEX 8000);
ENDDATA;

LET NOLKTYPES=6;

MODE LKTYPE(BYTE CHAR,LEN);

DATA LOOKTYPES;
   ARRAY(NOLKTYPES)LKTYPE TYPES:=(
      ('B',1),('C',1),('D',2),
      ('W',2),('F',2),('R',4));
ENDDATA;

%%%%%%%%%  TASK CONTROL  %%%%%%%%%


ENT PROC STOP(INT TASK);
   TASK:=TASK+NSTSK;
   IF TASK<=0 OR TASK > NTASKS THEN RRGEL(11); END;
    TASKLOCK := 1;  STAT(TASK):=STAT(TASK) LOR STOPP;
    CHANGE();                               % USER MAY BE STOPPING HIMSELF  %
                                            % CHANGE EFFECTIVELY CALLS      %
                                            % UNLOCK                        %
ENDPROC;


ENT PROC START(INT TASK);
   TASK := TASK+NSTSK;
   IF TASK <= 0 OR TASK>NTASKS THEN RRGEL(11); END;
   TASKLOCK := 1;  STAT(TASK) := STAT(TASK) LAND NOTSTOP;
   CHANGE();
                                            % CHANGE EFFECTIVELY CALLS      %
                                            % UNLOCK                        %
ENDPROC;



ENT PROC DELAY(INT TIME);
```

0-6-2

```
   IF TIME<=0 THEN RETURN END;
   INDEL(TIME);
   STAT(CURTEX) := STAT(CURTEX) LOR SUSP;
   CHANGE();
                                          % CHANGE EFFECTIVELY CALLS      %
                                          % UNLOCK();                     %
ENDPROC;


PROC INDEL(INT TIME);
   TASKLOCK := 1;
      BLOCK
   REF DELREC BPTR := ADEL,
              PTR,
               NPTR := FRPTR;             % TAKE NEW RECORD               %
    INT TIMEUP := TIME + NOW,
        TIMTST;
        FRPTR := FRPTR.NXT;
TRIAG:
          PTR := BPTR.NXT;
          IF PTR :=: ADEL
          THEN GOTO STKIN; END;
           TIMTST := TIMEUP - PTR.TIMUP;

                IF TIMTST > 0
                THEN BPTR := PTR;
                     GOTO TRIAG;
                END;

STKIN:
       NPTR.NXT   := PTR;                 % INSERT NEW RECORD            %
       BPTR.NXT   := NPTR;
       NPTR.TIMUP := TIMEUP;              % FILL IT UP                   %
       NPTR.TASK  := CURTEX;
   ENDBLOCK;
ENDPROC;



ENT PROC WAIT(INT EV);
   IF UNWAIT(CURTEX,EV) # 0
   THEN                                   % EVENT HAS OCCURRED           %
      IF TIMEOUT(CURTEX) = 0              % RESET TO ON A FIRST TIME     %
      THEN TIMEOUT(CURTEX) := -1;         % SUCCESS SO IT DIDN'T TIMEOUT %
      END;
      UNLOCK();
   ELSE EVFAC(CURTEX) := BYTE (EV+NSEV);
        CHAINWT(EVTOTSK(EV+NSEV),WTG);% WAIT FOR EVENT OR TIMEOUT        %
   END;
ENDPROC;


ENT PROC RESET(INT EV);
   UNWAIT(CURTEX,EV);
   UNLOCK();
ENDPROC;


ENT PROC WAITFOR(INT EV);
   UNWAIT(CURTEX,EV);                     % RESET                         %
   WAIT(EV);                              % WILL CALL CHANGE              %
ENDPROC;
```

```
ENT PROC LOCK();
   TASKLOCK := 1;
ENDPROC;
```

0-6-4

```
ENT PROC UNLOCK();
   % CALLED FROM S TASK AT S TASK STACK LEVEL, IT USES HARDWARE LOCKOUT%
   % FOR SHORT PERIODS, TO ALLOW THE EVENT QUEUE TO BE BUILT UP VIA    %
   % INTERRUPTS.IT CALLS SET, BUT SET WILL NOT GO TO CHANGE SINCE TASK %
   % LOCK IS SET. WHEN THE EVENT QUEUE IS EMPTY, IT CAN CALL CHANGE.   %

   INT EVENT;
EM: HLOCK();
   IF EVIN # EVOUT THEN
      EVOUT := (EVOUT+1) LAND NEVQS;
      EVENT :=  EVQ(EVOUT+1);
      HUNLOCK();
      SET(EVENT);
      HLOCK();
      IF EVIN=EVOUT THEN
         CHANGE();
      END;
      GOTO EM;
   END;
   TASKLOCK:=0;
   HUNLOCK();
ENDPROC;


ENT PROC SET(INT EV);

   INT WASLK :=TASKLOCK;
   INT EVDX  := EV + NSEV;
   IF EVDX <= 0 OR EVDX > NEVENTS THEN RRGEL(12); END;
   TASKLOCK := 1;                          % SET UP THE EVENT FOR ALL TASKS%
   FOR I := 1 TO NTDV16 DO
      EVBITS (I,EVDX) := HEX FFFF;
   REP;
   BLOCK
      REF BYTE TS := EVTOTSK(EVDX),TF;
      WHILE TS # 0
      DO    TF := WTCHN(TS);
            OUTDEL(TS);                   % TEST TIMEOUT AND PERHAPS EMPTY%
            UNWAIT(TS,EV);                % AND REMOVE TASK EVENT BITS,    %
            VAL TS := 0;                  % AND STAT CLEAR WTCHN ELEMENT   %
            TS := TF;                     % AND STEP ON                    %
      REP;
   ENDBLOCK;
   IF WASLK = 0 THEN CHANGE(); END;
ENDPROC;


ENT PROC SYSSTO(INT TIME);
   TIMEOUT(CURTEX):=TIME;
ENDPROC;


ENT PROC SYSRTO()INT;
   INT T:=TIMEOUT(CURTEX);
      TIMEOUT(CURTEX):=-1;
```

```
    RETURN(T);
ENDPROC;
```

(0-6-5)

```
PROC UNWAIT(INT TK,EV)INT;
% PRIVATE PROCEDURE TO REMOVE TASK EVENTS BIT AND TEST IT                %
   INT EVDX := EV + NSEV;
      IF EVDX <= 0 OR EVDX > NEVENTS
         THEN RRGEL(12);
      END;
   TASKLOCK := 1;
   BLOCK
   REF INT EVPTR := EVBITS(((TK-1)SRL 4) + 1,EVDX);
      INT MBIT:=(BITS(((TK-1)LAND 15)+1))LAND EVPTR;
      VAL EVPTR := EVPTR LAND NOT MBIT;
RETURN(MBIT);
ENDBLOCK;
ENDPROC;

ENT PROC ME()INT;
   RETURN(CURTASK);
ENDPROC;


%%%%%%%%%  FACILITY CONTROL  %%%%%%%%%


ENT PROC SECURE(INT FAC);
   FAC := FAC + NSFAC;
   IF FAC <= 0 OR FAC > NFAC THEN RRGEL(14); END;
   BLOCK
      REF BYTE F := FACILITY(FAC);
      REF INT T := TIMEOUT(CURTEX);
      IF F = CURTEX THEN RRGEL(16); END;
LP:
      TASKLOCK := 1;
      IF F = 0                              % FACILITY FREE                 %
      THEN VAL F := BYTE CURTEX;
           IF T = 0 THEN VAL T := -1 END;
   UNLOCK();
      ELSE EVFAC(CURTEX) := BYTE FAC;
           CHAINWT(FACTOTSK(FAC),WSEC);
           IF F # CURTEX AND T # 0 THEN
           GOTO LP
        END;
      END;
   ENDBLOCK;
ENDPROC;


ENT PROC RELEASE(INT FAC);
% RELEASE AWARDS THE FACILITY TO THE HIGHEST PRIORITY WAITING TASK      %
   FAC := FAC + NSFAC;                  % AND UNWAITS ANY STOPPED TASKS %
   IF FAC <= 0 OR FAC > NFAC THEN RRGEL(14); END;
   BLOCK
   REF BYTE TKPTR := FACILITY(FAC);
   INT P := HI := 0;
   IF TKPTR # CURTEX THEN RRGEL(15);   % CAN ONLY RELEASE OWN FACILITY %
   END;
   TASKLOCK := 1;
   BLOCK
   REF BYTE HPTR,TS := FACTOTSK(FAC),TF;
```

```
   WHILE TS # 0                                                    (0-6-6)
   DO TF := WTCHN(TS);
      IF STAT(TS)LAND STOPP # 0
      THEN OUTDEL (TS);                    % STOPPED TASKS UNWAITED FOR    %
      VAL TS := TF;                        % LATER TRY                      %
      VAL TF := 0;
      IF TS # 0 THEN
      TF := WTCHN(TS) END;
      ELSE IF PRIO(TS) > P
            THEN P := PRIO(TS);            % HIGHEST PRIORITY WAITING SO    %
            HPTR := TS;                    % FAR                            %
            HI := TS;
         END;
      END;
      TS := TF                             % TRY NEXT TASK                  %
   REP;
   IF P > 0
   THEN VAL HPTR := WTCHN(HI);             % CLEAR THE HIGHEST PRIORITY     %
                                          % WAITING TASK FROM THE WAIT     %
                                          % CHAIN FOR THIS FACILITY AND    %
        OUTDEL(HI);                       % JOIN UP THE CHAIN AND SET IT   %
   END;                                   % GOING, AND REMOVE IT FROM THE  %
                                          % DELAY LIST                     %
   VAL TKPTR := BYTE HI;                  % SET FACILITY EITHER FREE OR TO%
   CHANGE();                              % A NEW TASK                     %
ENDBLOCK;

ENDBLOCK;

ENDPROC;


PROC OUTDEL(INT TK);
% OUTDEL REMOVES ANY ENTRY TO THE DELAY CHAIN, AND SETS THE TASK NOT    %
% WAITING. EVFAC IS CLEARED. THE TASK WILL RESUME IN CHAINWT AND NEED   %
% NOT CLEAR ANY DATA. IT USES TASK INDEX.                               %

   REF INT T := TIMEOUT(TK);
      STAT(TK) := STAT(TK)LAND STOPP;  % SET TASK NOT WAITING            %
      EVFAC(TK) := 0;
   IF T > 0                            % IN THIS CASE A RECORD STILL    %
   THEN                                % EXISTS IN THE DELAY LIST       %
   BLOCK                               % EXTRACT IT                     %
      REF DELREC BPTR := ADEL,
                 PTR := ADEL.NXT;
                 WHILE PTR :#: ADEL
                 DO
                    IF PTR.TASK = TK
                    THEN VAL T := PTR.TIMUP - NOW;
                         BPTR.NXT := PTR.NXT;
                         PTR.NXT := FRPTR;
                         FRPTR := PTR;
                         GOTO RET;
                      END;
                      BPTR := PTR;
                      PTR := PTR.NXT;
                    REP;
      ENDBLOCK;
   END;
RET:

ENDPROC;
```

```
PROC CHAINWT(REF BYTE TKPTR, INT STWT);
% LOCAL PROCEDURE TO HANDLE WTCHN AND DELAY CHAIN FOR WAIT AND SECURE  %

   INT T := TIMEOUT(CURTEX);
   IF T > 0
   THEN INDEL(T);                          % PUT TASK IN DELAY CHAIN IF    %
   END;                                    % TIMEOUT SET                   %
   IF T # 0                                % DROP THROUGH ON ZERO TIMEOUT  %
   THEN
   STAT(CURTEX):=STAT(CURTEX)              % WAIT, OR WTSEC STATUS         %
      LOR BYTE STWT;
      WTCHN(CURTEX) := TKPTR;              % PUT CURTEX INTO WAIT CHAIN    %
      VAL TKPTR     := BYTE(CURTEX);
      CHANGE();                           % WAIT TILL ACTIVATED BY RELEASE%
   ELSE EVFAC(CURTEX) := 0;                % SET STIM, OR T.O.             %
   UNLOCK();
   END;

ENDPROC;


ENT PROC CLEANUP();
   FOR I:=1 TO NFAC DO
      IF FACILITY(I)=CURTEX THEN RELEASE(I - NSFAC) END;
   REP;
ENDPROC;


%%%%%%%%%  SECURE RELEASE ADDITIONS  %%%%%%%%%

ENT PROC TSTSCR(INT FAC)INT;
   % RESULT IS 0 IF CURTEX HAS NOT SECURED FAC 1 IF IT HAS                 %
   FAC := FAC + NSFAC;
   IF FAC <= 0 OR FAC > NFAC THEN RRGEL(14); END;
   RETURN(IF FACILITY(FAC)=CURTEX THEN 1 ELSE 0 END);
ENDPROC;


ENT PROC TWAIT(INT EV,TIME,LABEL FAILEXIT);
   % WAITS FOR EVENT EV FOR TIME/50 SECS                                   %
   % IF EVENT IS NOT SET THEN GOES TO FAILEXIT                             %
   SYSSTO(TIME);
   WAIT(EV);
   IF SYSRTO()=0 THEN GOTO FAILEXIT END;
ENDPROC;


ENT PROC TSECURE(INT FAC,TIME,LABEL FAILEXIT);
   % TRIES TO SECURE FACILITY FAC FOR TIME/50 SECS                         %
   % GOES TO FAILEXIT IF UNSUCCESSFUL                                      %
   SYSSTO(TIME);
   SECURE(FAC);
   IF SYSRTO()=0 THEN GOTO FAILEXIT END;
ENDPROC;


%%%%%%%%%  ERROR PRINTING  %%%%%%%%%
```

```
ENT PROC ERPRIN();                          % TASK BASE PROCEDURE          %
   INT Y;
L:
   STOP(ERPTNO);                            % WAIT TILL RESTARTED BY RRGEL  %

% THE ERROR PRINTING HAS BEEN COMMENTED OUT. WHEN AN UNRECOVERABLE ERROR %
% OCCURS, SMTCOMS OUTPUTS A MEDIA ERROR MESSAGE OF THREE BYTES.          %

   IF ECT#0 THEN
%      OUT:=ERROUT;                                                      %
%      TWRT("#NL,BELL#ERROR "); IWRT(ERNO);                              %
%      OUT(SP); TIMDAT(-1);                                              %
%      IF TASKNO#0 THEN                                                  %
%         TWRT(" TASK "); IWRT(TASKNO);                                  %
%         IF LINENO >= 0 THEN TWRT(" ON LINE "); IWRT(LINENO); END;      %
%      ELSE                                                              %
%         TWRT(" IN H-TASK");                                            %
%      END;                                                              %
%      OUT(NL);                                                          %
%      FOR I:=1 TO 9 DO OWRT(REGS(I)); OUT(SP) REP;                      %
%      TWRT("#NL#STACK#NL#");                                            %
%      FOR I := 1 TO NSTCON DO OWRT(STKVALS(I));                         %
%         OUT(IF I LAND 7 = 0                                            %
%            THEN NL ELSE SP                                             %
%         END);                                                          %
%      REP;                                                              %
%         OUT(NL);                                                       %
      LOCK(); Y:=ECT; ECT:=0; UNLOCK();
%      IF Y#1 THEN TWRT("#NL#ER DATA LOST#NL#") END;                     %
   END;
   GOTO L;
ENDPROC;


%%%%%%%%% CONTROL A EVENT RESPONSE TASK %%%%%%%%%

LET GO      = HEX 4F47    ;
LET STP     = HEX 5053    ;
LET LK      = HEX 4B4C    ;
LET SE      = HEX 4553    ;
LET TD      = HEX 4454    ;
LET ND      = HEX 444E    ;

EXT DATA INSTDATA;
   LABEL NIXL;
   BYTE NXCH;
ENDDATA;


ENT PROC INSTRUCT();                        % TASK BASE PROCEDURE          %
   INT X,Y,ACTION;
   INT NI,TYPTR,TYPLEN;
   BYTE TYPE;
   REAL YR;

   IN:=CHIN;                                % INITIALISE I/P FOR LOOK AHEAD %
ENTER:                                      % RESTART POINT                %
   OUT:=CTLAOUT;
   CLEANUP();                               % IN CASE OF ERROR ENTRY        %
   ERL:=ENTER;  NIXL:=NIX;  ERP:=GOTOLBL;
```

```
WAIT(CTLAEV);                          % WAIT FOR CTLAEV                    %

% RESPOND TO PRESSING OF CTL A KEY                                          %
NXCH:=0;
TWRT("#NL#CTLA#ENQ#");
ACTION:=MNPACK();                      % RETURNS 2-CHAR MNEMONIC PACKED     %
                                       % INTO INT                           %
IF ACTION=TD THEN
   SPS(3);
   TIMDAT(-1);
   GOTO OK
END;
IF ACTION=ND THEN SETDAT(); GOTO ENTER END;

IF IN()#',' THEN GOTO NIX END;         % PARAMETER SHOULD FOLLOW            %

IF ACTION=LK THEN                      % CORE LOCATION LOOK                 %
   NI:=1; TYPE:='W';                   % DEFAULT NO OF ITEMS AND TYPE       %
ABSADR:
   X:=0;
NEWADR:
   X:=X+ZREAD();                       % NEW LK ADDRESS                     %
READNOITYPE:
   NIXL:=NIX;
   WHILE TERMCH#EOM DO
      IF TERMCH#',' THEN GOTO NIX END;
      BLOCK BYTE T:=NEXT();
         NIXL:=READTYPE;               % TO READTYPE IF ZREAD FAILS         %
         NI:=ZREAD();
         GOTO NT;
      READTYPE:
         TYPE:=T; TERMCH:=IN();
      NT:
         NIXL:=NIX;
      ENDBLOCK;
   REP;
   FOR I:=1 TO NOLKTYPES DO
      IF TYPES(I).CHAR=TYPE THEN
         TYPTR:=I; TYPLEN:=TYPES(I).LEN; GOTO TYPOK;
      END;
   REP;
   GOTO NIX;                           % ILLEGAL TYPE                      %
TYPOK:
   IF TYPLEN#1 THEN X:=X LAND OCT 177776 END;
   TO NI DO
      REF INT A:=ITORI(X LAND OCT 177776);
      IF NI#1 THEN OUT(NL) END;
      OUT('@'); OWRT(X); OUT('=');

      SWITCH TYPTR OF BB,CC,II,WW,FF,RR;
                                       % SWITCH CANNOT FAIL                %
   BB:
   CC:
      BLOCK
         BYTE B := ITORB(X);
         OWRT(B);
         IF TYPTR=2 THEN OUT(','); OUT(B) END;
         GOTO UPDX;
      ENDBLOCK;
   II:
      IWRT(A);
      GOTO UPDX;
```

```
      WW:
         OWRT(A);
         GOTO UPDX;
      FF:
         BLOCK
            REF FRAC RF:=ITORF(X);
            RWRTF(REAL RF,0,7);
            GOTO UPDX;
         ENDBLOCK;
      RR:
         BLOCK
            REF REAL RR:=ITORR(X);
            RWRTF(RR,0,7);
         ENDBLOCK;
      UPDX:
         X:=X+TYPLEN;                    % ADVANCE ADDRESS FOR NEXT LK   %
      REP;

      IF NI#1 THEN GOTO NXTLK END;       % NO REPLACEMENT VALUE          %
      NXCH:=0;
      X := X - TYPLEN;
      TWRT(" :=#ENQ#");
      IF NEXT()=EOM THEN GOTO REPOK END;
      SWITCH TYPTR OF RBB,RCC,RII,RWW,RFF,RRR;
                                         % SWITCH CANNOT FAIL           %
   RBB:
   RII:
   RWW:
      Y:=ZREAD(); GOTO CHKRD;
   RCC:
      Y:=IN(); TERMCH:=IN(); GOTO CHKRD;
   RFF:
      Y:=INT(FREAD() SLA 15);            % PRETEND FRAC IS AN INT PATTERN%
      GOTO CHKRD;                        % HAS NOT BEEN CHANGED BY THIS  %
   RRR:
      YR:=RREAD();
   CHKRD:
      IF TERMCH#EOM THEN GOTO NIX END;


SWITCH TYPLEN OF ONEB,TWOB,REPOK,FOURB;% 3 NOT POSSIBLE                  %
                                         % SWITCH CANNOT FAIL           %
   ONEB:
      BLOCK
         REF BYTE RB := ITORB(X);
            VAL RB := BYTE(Y);
         GOTO REPOK;
      ENDBLOCK;
   TWOB:
      BLOCK
         REF INT RI:=ITORI(X);
         VAL RI:=Y;
         GOTO REPOK;
      ENDBLOCK;
   FOURB:
      BLOCK
         REF REAL RR:=ITORR(X);
         VAL RR:=YR;
      ENDBLOCK;
   REPOK:
      X:=X+TYPLEN;
      TWRT(" OK");
```

0-6-10

```
   NXTLK:
      NXCH:=0;
      TWRT("#NL(2)#CTLA LK#ENQ#");
      IF NEXT()=EOM THEN TWRT(" EXIT");
         GOTO OK END;                        % FINISHED                    %
      IF NXCH#'+' AND NXCH#'-' THEN GOTO ABSADR END;
      NIXL:=READNOITYPE;                     % WREAD ERROR MEANS SPECIAL    %
                                             % + OR - CASE                 %
      GOTO NEWADR;
   END;

   X:=IREAD();                               % SHOULD BE AN INTEGER PARAMETER%
   IF TERMCH#EOM THEN GOTO NIX END;

   IF ACTION=GO THEN START(X);               % START TASK                   %

   ELSEIF ACTION=STP THEN STOP(X);           % STOP TASK                    %

   ELSEIF ACTION=SE THEN SET(X);             % SET EVENT                    %

   ELSE GOTO NIX;                            % UNKNOWN MNEMONIC             %
   END;
OK:
   TWRT(" OK#NL#");   GOTO ENTER;
NIX:
   TWRT(" NO#33,NL#");   GOTO ENTER;

ENDPROC;

ENT PROC MNPACK()INT;
   BYTE CHAR:=IN();
   RETURN(CHAR LOR (IN() SLL 8));
ENDPROC;


PROC NEXT()BYTE;
   IF NXCH=0 THEN NXCH:=CTLAIN() END;
   RETURN(NXCH);
ENDPROC;

PROC CHIN()BYTE;
   BYTE B:=IF NXCH=0 THEN CTLAIN() ELSE NXCH END;
   NXCH:=0;
   RETURN(B);
ENDPROC;

PROC ZREAD()INT;
   INT N:=0;

NEXT:
   N:=N+WREAD();
   IF TERMCH='+' OR TERMCH='-' THEN
      NXCH:=TERMCH;
      GOTO NEXT;
   END;
   RETURN(N);
ENDPROC;

PROC GOTOLBL(INT N);                         % CTLA TASK ERROR PROCESSING   %
   GOTO NIXL;
ENDPROC;
```

```
%%%%%%%%%  DEFAULT PROCEDURES  %%%%%%%%%

ENT PROC DFERP(INT N);
ENDPROC;

ENT PROC DEFIN()BYTE;
   RETURN(OCT 200);
ENDPROC;

ENT PROC DEFOUT(BYTE B);
ENDPROC;

ENT PROC RRNUL();
ENDPROC;

ENT PROC FBPROC();   %%%%% FBTASK BASE PROCEDURE %%%%%
L: GOTO L;
ENDPROC;


%%%%%%%%%  I/O FORMAT PROCEDURES  %%%%%%%%%


%%%%% TIMDAT %%%%%

ENT PROC TIMDAT(INT T);
   INT H,MI,S,D,MO,Y;

   LOCK();                                  % SAMPLE TIME AND DATE          %
                                            %INDIVISIBLY                     %
      H:=HOURS;  MI:=MINS;  S:=SECS;
      D:=DAYS;  MO:=MONTHS;  Y:=YEARS;
   UNLOCK();

   IF T<=0 THEN
      PUT2(H,':');
      PUT2(MI,':');
      PUT2(S,IF T<0 THEN' ' ELSE 0 END);
  END;
   IF T#0 THEN
      PUT2(D,'/');
      PUT2(MO,'/');
      PUT2(Y,0);
   END;
ENDPROC;

ENT PROC PUT2(INT VLU,BYTE SEP);
   OUT(BYTE(VLU:/10+'0'));
   OUT(BYTE(VLU MOD 10+'0'));
   IF SEP#0 THEN OUT(SEP);  END;
ENDPROC;




%%%%%%%%%%% IREAD %%%%%%%%%%%

ENT PROC IREAD () INT;
   INT SIGN;
```

```
   RETURN (BUILD(FIRSTCH(SIGN),10,SIGN,101));
ENDPROC;
```

0-6-13

```
%%%%% IWRT %%%%%

ENT PROC IWRT(INT X);
   IF X>=0 THEN X:=-X ELSE OUT('-') END;
   IF X<=-10 THEN IWRT(X:/-10) END;
   OUT(BYTE(-(X MOD -10))LOR '0');
ENDPROC;


%%%%% TWRT %%%%%
ENT PROC TWRT(REF ARRAY BYTE X);
   FOR I:=1 TO LENGTH X DO OUT(X(I)) REP;
ENDPROC;


%%%%% OWRT %%%%%

ENT PROC OWRT(INT X);
   OUT('''');
   FOR I:=15 BY -3 TO 0 DO
      OUT(BYTE((X SRL I)LAND 7)LOR '0');
   REP;
ENDPROC;




%%%%%%%%%%% OREAD %%%%%%%%%%%

ENT PROC OREAD () INT;

   INT SIGN, CH := FIRSTCH(SIGN);

   RETURN (BUILD(IF CH = '''' THEN IN()
      ELSE CH + 256 END,8,SIGN,104));  % WILL FAIL,BUT TERMCH WILL      %
                                       % BE BYTE CH                     %

ENDPROC;




%%%%%%%%%%% WREAD %%%%%%%%%%%

ENT PROC WREAD () INT;
   INT SIGN, CH := FIRSTCH(SIGN),RADIX := 10;
   IF CH = ''''
   THEN RADIX := 8; CH := IN();
   END;
   RETURN (BUILD(CH,RADIX,SIGN,105));
```

```
ENDPROC;
```

0-6-14

```
%%%%%  NLS  %%%%%

ENT PROC NLS(INT N);
   TO N DO OUT(NL) REP;
ENDPROC;


%%%%%  SPS  %%%%%

ENT PROC SPS(INT N);
   TO N DO OUT(SP) REP;
ENDPROC;




%%%%%%%%%%%%%%%%% RWRTF %%%%%%%%%%%%%%%%%

ENT PROC RWRTF(REAL Y, INT M,N);

  ST:
   BLOCK
   REAL ROUND := 0.5, LIM, X := Y;
   INT EXP := 0, ABSEXP,PLACE := 1,INTPT;
   BYTE SIGN := SP;

   IF X < 0.0 THEN X := -X; SIGN := '-' END;

      TO N DO ROUND := ROUND/10.0 REP;
         IF M # 0 THEN
         X := X + ROUND;
      END;

   WHILE X >= 10.0 DO X := X/10.0; EXP := EXP + 1 REP;
   IF M > EXP THEN
      PLACE := EXP + 1;
      TO M - PLACE DO OUT(SP) REP;
  ELSE
   IF M # 0 THEN N := M + N; M := 0; GOTO ST; END;
   LIM := 1.0 - ROUND;
   WHILE X < LIM AND X # 0.0 DO X := X*10.0; EXP := EXP - 1 REP;
   X := X + ROUND;
END;
   OUT(SIGN);
   FOR I := 1 TO PLACE + N DO
      INTPT := INT(X - 0.5);
      OUT(BYTE INTPT LOR '0');
      X := (X - INTPT)*10.0;
      IF I = PLACE AND N # 0 THEN OUT('.') END;
   REP;
   IF M = 0 THEN
      ABSEXP := ABS EXP;
```

```
      OUT('E');
      OUT(IF EXP >= 0 THEN '+' ELSE '-' END);
      IF ABSEXP < 10 THEN OUT ('0') END;
         IWRT(ABSEXP);
   END;
ENDBLOCK;
ENDPROC;



%%%%%%%%%% FREAD %%%%%%%%%%

LET MAXFRAC = 32767.0B-15;              % LARGEST 16 BIT FRACTION      %


ENT PROC FREAD()FRAC;

   REAL N := RREAD();                   % READ AS A REAL               %

   IF N > 1.0 OR N < -1.0               % TOO BIG, OR TOO SMALL?       %
   THEN IOFLAG := 1;
        ERP(100);
        N := 0.0;
 ELSEIF N = 1.0 THEN RETURN (MAXFRAC)   % JUST RIGHT                   %
   END;

   RETURN (FRAC (N));

ENDPROC;



%%%%%%%%%% RREAD %%%%%%%%%%

ENT PROC RREAD() REAL;

   REAL  B := 1.0, R := 0.0;
   INT  EXP, SIGN, SGNEXP,
       A := 10, STARTED := 0,

   CH := FIRSTCH(SIGN);                 % SIGN AND LAYOUT              %

   WHILE  CH >= '0' AND CH <= '9'       % BUILD THE NUMBER            %
   DO STARTED := 1;
      IF A = 1 THEN B := B/10.0 END;
      R := R*A + B*(CH - '0');
NXCH:
      CH := IN();
   REP;

   IF CH='.' AND STARTED=1 AND A = 10   % CHANGE TO FRPT              %
   THEN STARTED := 0;
        A := 1;
        GOTO  NXCH;
   END;
   TERMCH := BYTE CH;

   TO IF STARTED = 1 AND CH # 'E'
```

0-6-16

```
      THEN 0
      ELSE BUILD (IF STARTED = 1 THEN FIRSTCH(SGNEXP)
            ELSE CH + 256                  % WILL FAIL                    %
            END,
            10, 1,                         % ABSOLUTE VALUE               %
            102)
         END
        DO
            R := R* IF SGNEXP > 0 THEN 10.0 ELSE 0.1 END;
         REP;

RETURN (R * SIGN);


ENDPROC;

   % SET DATE AND TIME, USED BY CTLA TASK AND STARTUP JOB                  %

LET ENQ=5;
LET EOM=3;

ENT PROC SETDAT();
   INT DYS,MNTHS,YRS,HRS,MNS;

   IOFLAG:=0;
   TWRT(" DATE#ENQ#");
   DYS:=CREAD(1,31);
   MNTHS:=CREAD(1,12);
   YRS:=CREAD(0,99);
   TWRT(" TIME#ENQ#");
   HRS:=CREAD(0,23);
   MNS:=CREAD(0,59);
   IF IOFLAG=0 THEN
      LOCK();
      DAYS:=DYS; MONTHS:=MNTHS; YEARS:=YRS;
      HOURS:=HRS; MINS:=MNS; SECS:=0;
      UNLOCK();
      TWRT(" OK#NL#");
   ELSE TWRT(" NO#33,NL#") END;
ENDPROC;

ENT PROC CREAD(INT LO,HI)INT;
   INT X:=IREAD();
   IF X<LO OR X>HI THEN
      IOFLAG:=3;
   END;
   RETURN(X);
ENDPROC;



%%%%%%%%%%% FIRSTCH %%%%%%%%%%%

PROC FIRSTCH (REF INT SIGN) INT;
% A PRIVATE PROCEDURE TO DETECT FIRST NON LAYOUT CHARACTER AND SIGN       %
      BYTE CHAR;
      VAL SIGN := 1;

RDC:
      CHAR := IN();
         IF CHAR = SP OR CHAR = NL OR CHAR = TAB THEN
```

```
         GOTO RDC
      END;

   IF CHAR = '-'
   THEN VAL SIGN := -1;
        GOTO GETC
   END;
   IF CHAR = '+'
   THEN
GETC:
   CHAR := IN();
   END;

RETURN (CHAR);

ENDPROC;



%%%%%%%%%% BUILD %%%%%%%%%%

PROC BUILD (INT CH,RADIX,SIGN,ERNUM) INT;
% BUILDS ANY RADIX TO 10, USES SUPPLIED SIGN                    %
   INT STARTED := N := 0;

   BLD:

      TERMCH := BYTE CH;

      CH := CH - '0';

      IF CH >= 0 AND CH < RADIX
      THEN STARTED := 1;
           N := N * RADIX  -CH;
           CH := IN();
           GOTO BLD
      END;
      IF STARTED = 0 THEN IOFLAG := 1;
      ERP(ERNUM);
   END;
   RETURN(IF SIGN > 0 THEN -N ELSE N END);

ENDPROC;
```

0-6-17

```
TITLE
      SMT OPERATING SYSTEM
      ( WITH NEGATIVE TASK NUMBERS )
      SYSTEM DATA BRICKS AND STACKS (READ/WRITE)
      **** MODULE SMTB3 ****
      SMT 3(18) 13-04-1981;

OPTION (1);


LET NTASKS     =  22;                    %                              %
LET NTDV16     =   2;                    %                              %
LET NSEV       =  16;                    %                              %
LET NEVENTS    =  32;                    %                              %
LET NFAC       =  32;                    %                              %
LET NSFAC      =  16;                    %                              %
LET EVQLEN     =  16;                    %                              %
LET NEVQS      =  15;                    %                              %
LET INVEVS     = -16;                    %                              %
LET STOPP      =  OCT 100;               %                              %
LET NSTCON     =  16;                    %                              %

EXT PROC () BYTE DEFIN;
EXT PROC (BYTE) DEFOUT;

MODE DELREC ( INT TIMUP, TASK, REF DELREC NXT );
MODE TASKLINK ( REF ARRAY STACK TKS,
                REF ARRAY PROC() TKP,
                REF ARRAY BYTE TKPR,TKST,
                PROC () UINIT,UPF);

%%%%%%%%% STACKS FOR SYSTEM TASKS %%%%%%%%%

ENT STACK HSTK 160;                      % HARDWARE TASK                 %
ENT STACK FBSTK 70;                      % FALL-BACK TASK                 %
ENT STACK INSTK 220;                     % CTLA TASK                      %
ENT STACK SYSTACK 160;                   % SYSTEM STARTUP AND 'SET' TASK  %
ENT STACK CLKSTK 160;                    % CLOCK SERVICING TASK           %
ENT STACK ERRSTK 160;                    % ERROR PRINT TASK               %

ENT DATA TASKDATA;
   INT CURTASK;                          % NO. OF CURRENTLY ALIVE S-TASK  %
   INT CURTEX;                           % ARRAY INDEX OF CURRENTLY ALIVE%
                                         % S-TASK                        %
   INT TASKLOCK := 0;                    % NON-ZERO IF TASK CHANGING IS   %
                                         %LOCKED OUT                     %
   INT NXTCUR;                           % TEMPORARY STORE                %
   INT EVIN := 0;                        % INPUT POINTER TO EVENT QUEUE   %
   INT EVOUT := 0;                       % O/P POINTER FROM EVENT QUEUE   %
   DELREC ADEL;
   REF DELREC FRPTR;
   BYTE HIPRI := OCT 340,LOPRI := 0;     % HI AND LO PRIORITY FOR LSI     %
   ARRAY(NTASKS)INT TIMEOUT := (-1(NTASKS));% TIMEOUT TO BE USED          %
   ARRAY(NTASKS)DELREC DEL;
   ARRAY(NTASKS)STACK CELL;              % STACK ADDRESSES                %
   ARRAY(NTASKS)BYTE EVFAC:=(0(NTASKS)),% AWAITED EVNT/FAC NO.           %
               STAT := (STOPP(NTASKS)),% TASK STATE                      %
               PRIO := (1(NTASKS))    ,% TASK PRIORITY                   %
               WTCHN := (0(NTASKS))   ;% CHAINED LIST FOR WAITING FOR    %
                                       % EVENT OR FAC                    %
   ARRAY(NTDV16,NEVENTS)INT EVBITS := ((0(NEVENTS))(NTDV16));
```

```
                                        % EVENT SET BITS                    %
   ARRAY(NTASKS)REF BYTE STATADS := (STAT(1)(NTASKS));
   % ADDRESSES OF STATUS BYTES OF TASKS IN PRIORITY ORDER                    %
   % NOTE THAT THE FALL BACK TASK IS ALWAYS IN A GO STATE                    %
   ARRAY(EVQLEN)INT EVQ;                % THE EVENT QUEUE                   %
   ARRAY(NFAC)BYTE FACILITY:=(0(NFAC)),FACTOTSK:=(0(NFAC));% FACILITIES%
                                        % HEAD OF THE TASKCHAIN WAITING %
                                        % FOR EACH FACILITY                 %
   ARRAY(NEVENTS)BYTE EVTOTSK:=(0(NEVENTS));
                                        % HEAD OF THE TASKCHAIN WAITING %
                                        % FOR EACH EVENT                    %
ENDDATA;


ENT DATA TIMEDATA;
   INT   NOW,                           % CYCLIC 1/50 SEC COUNTER           %
      SECSNOW,                          % CYCLIC 1 SECOND COUNTER           %
      MINSNOW,                          % CYCLIC MINUTES COUNTER            %
      TCOUNT,                           % REALTIME CLOCK 50TH SEC(0-49)     %
      SECS   := 30,                     % SECONDS (0 TO 59)                 %
      MINS   := 49,                     % MINUTES (0 TO 59)                 %
      HOURS  := 14,                     % HOURS   (0 TO 23)                 %
      DAYS   := 13,                     % DAYS    (0 TO 31)                 %
      MONTHS := 04,                     % MONTHS  (0 TO 12)                 %
      YEARS  := 81;                     % YEARS   (0 TO 99)                 %
ENDDATA;


ENT DATA CLKDATA;
   INT TICK := 0;                       % STORES CLOCKTICKS TILL            %
ENDDATA;                                % CLOCKTASK CATCHES UP              %


ENT DATA REGDATA;
   ARRAY(9)INT REGS;
   ARRAY(NSTCON)INT STKVALS;
   INT ECT,ERNO,TASKNO,LINENO := -1;
ENDDATA;


ENT DATA TRAPDATA;
   LABEL TASKEXIT;
   REF TASKLINK T;
   INT UR1,ERNUM,UPS;
ENDDATA;

ENT DATA PWFDATA;
   INT PWFLAG;
ENDDATA;

ENT DATA SYSIODATA;
   PROC () BYTE CTLAIN := DEFIN;        % CTLA TASK 'IN' PROCEDURE          %
   PROC (BYTE) CTLAOUT := DEFOUT;       % CTLA TASK 'OUT' PROCEDURE         %
   PROC (BYTE) ERROUT  := DEFOUT;       % ERROR PRINT 'OUT' PROCEDURE       %
ENDDATA;

ENT DATA INSTDATA;                      % DATA USED BY CTLA TASK            %
   LABEL NIXL;                          % ERROR RE-ENTRY LABEL              %
   BYTE NXCH;
ENDDATA;
```

```
        .PSECT  RTLCTL
;CONTROL ROUTINES FOR RTL/2
;
; OPTION NO. * EIS M/C * REAL VARIABLES * "EI" OPTION * "FP" OPTION
;            *         *    SUPPORTED   * ALWAYS USED * ALWAYS USED
;*************************************************************************
;            *         *                *              *
;    (5)     *    YES  *       YES      *       NO     *       NO
;            *         *                *              *
        .TITLE CONTROL ROUTINES RTL/2 SET 05
ZHWMD:
ZFLPT:
        .PAGE
        .SBTTL CONFIGURATION DETAILS
; CONTROL ROUTINES FOR RTL/2 - OPTIONAL EIS AND FLOATING POINT
;
;               DOS RELEASE 2 VERSION (AFTER RSX-11 VERSION) FOR
;               NEW COMPILER IN-LINE CAPABILITY AND
;               11/45 FLOATING POINT INSTRUCTIONS
;               DOS RELEASE 3 VERSION   21-JULY-77  CID
;               NEW BASIC FLOATING POINT ROUTINES (EX RELEASE 215)
;               EIS VERSION OF R21 & R22 INCORPORATED
;               11/45 FPOFLO ROUTINE MODIFIED TO USE X1 (PRESERVING X4)
;
;               RSX11-M VERSION 5 FROM DOS 13/9/77 GW
;
;
;
;  IF SYMBOL ZFLPT IS DEFINED FLOATING AND FIXED POINT
;                             ROUTINES ARE ASSEMBLED ELSE
;                             FIXED POINT ONLY
;
;  IF SYMBOL ZHWMD IS DEFINED USE IS MADE OF HARDWARE MULTIPLY
;                             AND DIVIDE INSTRUCTIONS
;                             FOR INTEGER AND FRAC
;
;  IF SYMBOL ZEISIL IS DEFINED FIXED POINT MULTIPLY,DIVIDE,SHIFT
;                              ROUTINES ARE NOT ASSEMBLED
;
;  IF SYMBOL ZFPIL IS DEFINED  FLOATING POINT ROUTINES CONSIST OF TYPE
;                              CONVERSIONS ONLY - ADD,SUB,MUL,DIV ARE
;                              OMITTED
;
;  IF SYMBOL Z45FP IS DEFINED FLOATING POINT ROUTINES USE 11/45
;                             FLOATING POINT PROCESSOR INSTRUCTIONS
;
;
;  ROUTINES OMITTED HAVE NO GLOBAL DECLARATION - PROGRAMS WITH
;        JSR R2,RXX WILL FAIL TO LINK CORRECTLY, PROGRAMS WITH
;        TRAPS TO EXCLUDED ROUTINES WILL CAUSE RRGEL CALL
;
;
;
; THE STACK USAGES GIVEN BELOW ARE IN THE FORM
; (PARAMETERS+LINK ADDRESS) + WORKSPACE   E.G.   4+3
;
;
        .PAGE
        .SBTTL GLOBALS & ASSIGNMENTS
.IF DF Z45FP
        AC0=%0
```

```
        AC1=%1
.ENDC                   ;(Z45FP)

         X1=%1
         X2=%2
         X3=%3
         X4=%4
         X5=%5
         X6=%6
         X7=%7
;
;
.GLOBL STKUSG                           ; ***SMT
.GLOBL R00,R23,R24,SMT...               ; ***SMT

.GLOBL RRGEL
.GLOBL CNTRTN
.GLOBL R01,R02,R03,R04,R25,R17,R18,R19,R20,R21,R09
.IF NDF ZEISIL
.GLOBL R05,R06,R07,R08,R10,R11,R12,R13
.IFTF                   ;(ZEISIL)
.GLOBL R14,R15,R16,R22

.ENDC                   ;(ZEISIL)
.IF DF ZFLPT
.GLOBL R30,R31,R32,R33,R34,R35,R36,R37,R38,R39
.IF NDF ZFPIL
.GLOBL R26,R27,R28,R29
.ENDC                   ;(ZFPIL)
.IFTF                   ;(ZFLPT)
;
;
SKLO=2          ; ADDRESS OF BOTTOM OF USABLE STACK FOR R01 CHECKING
        STKFLO=1
        LABERR=2
        ARRERR=4
;
.IFF                    ;(ZFLPT)
;
MAXTRAP=50.             ; MAXIMUM CONTROL ROUTINE NO SUPPORTED
                        ; ( *2 FOR RSX *** )
                        ;    BY TRAP HANDLER
.IFT                    ;(ZFLPT)
MAXTRAP=78.             ; MAXIMUM CONTROL ROUTINE SUPPORTED
                        ; ( *2 FOR RSX  ***)
                        ;    BY TRAP HANDLER
        .PAGE
        .SBTTL TRAP HANDLER
.IFTF                   ;(ZFLPT)
;
; TRAP HANDLER - TRAP 1 TO TRAP MAXTRAP ARE PROCESSED AS CALLS
;  TO CORRESPONDING CONTROL ROUTINES, APPEARING TO ROUTINES AS
;        JSR   X2,RNN
;

        .ASECT
        .=34
        .WORD   CNTRTN,340
        .PSECT  RTLCTL
CNTRTN:
;
; FROM HERE TO TRP.2 MODIFIED FOR RSX
```

(0-8-3)

```
;
; THIS IS BASICALLY THE RSX-11M VERSION, WITH THE ACTION OF AN SST
; HANDLER SIMULATED IN THE NEXT FEW INSTRUCTIONS...

        MOV     (SP),-(SP)          ; COPY PC VALUE
        SUB     #2,(SP)             ; BACK OFF TO TRAP INSTR ADDRESS (PC-2)
        MOV     @(SP)+,-(SP)        ; PICKUP BOTTOM BYTE = TRAP OPERAND
        BIC     #177400,(SP)
        ASL     (SP)                ; *2 FOR WORD OFFSET INTO TRAP TABLE
;
        CMP     (6),#MAXTRAP        ;CONTROL RTN CALL
        BGT     TRP.1
        MOV     4(SP),177776        ; RESTORE USER'S PSW (**SMT**)
        MOV     X2,4(6)             ; X2 AS FOR JSR 2,RNN
        MOV     2(6),X2             ; LINK INTO X2
        MOV     (6)+,(6)            ; TRAP OPERAND TO STACK
        ADD     #TRP.2-2,(6)        ; ADDR OF ROUTINE
         MOV    @(6)+,X7                   ; SIMULATE JSR 2,RNN FROM USER
;    N.B. THE ABOVE INSTRUCTION IS A SUBSTITUTE FOR JMP (6)+ WHICH
;    VARIES IN BEHAVIOUR BETWEEN DIFFERENT MODELS OF PDP11
;
TRP.2:

R01.0,R02.0,R03.0,R04.0
.IF DF ZEISIL
TRP.1,TRP.1,TRP.1,TRP.1
.IFF                        ;(ZEISIL)
R05.0,R06.0,R07.0,R08.0
.IFTF                       ;(ZEISIL)
R09.0
.IFT                        ;(ZEISIL)
TRP.1,TRP.1,TRP.1,TRP.1,R14.0,R15.0,R16.0

.IFF                        ;(ZEISIL)
R10.0,R11.0,R12.0,R13.0,R14.0,R15.0,R16.0
.IFTF                       ;(ZEISIL)
R17.0,R18.0,R19.0,R20.0,R21.0
.IFT                        ;(ZEISIL)
TRP.1
.IFF                        ;(ZEISIL)
R22.0
.ENDC                       ;(ZEISIL)
R23,R24,R25.0               ; (**SMT**)
.IFT                        ;(ZFLPT)
.IF DF ZFPIL
TRP.1,TRP.1,TRP.1,TRP.1
.IFF                        ;(ZFPIL)
R26.0,R27.0,R28.0,R29.0
.ENDC                       ;(ZFPIL)
R30.0,R31.0,R32.0,R33.0,R34.0,R35.0,R36.0,R37.0,R38.0,R39.0
.ENDC                       ;(ZFLPT)
TRP.1:  JMP     @R00+6                     ; SVC PROC CALL (**SMT**)
        .PAGE
        .SBTTL STACK ADMINISTRATION
;
;
; PROCEDURE ENTRY, CHECKS FOR STACK OVERFLOW
;
R01:
R01.0:   MOV    X6,X3                      ; COPY X6
         SUB    (2)+,X3                    ; COMPUT MAX STACK USAGE
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        CMP    X3,STKUSG+0(0)              ; RECORD LOWEST ADDRESSED STK ***RS
        BHIS   R01.2
        MOV    X3,STKUSG+0(0)              ; ***SMT
R01.2:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        CMP    X3,SKLO(0)                  ; COMPARE TO LOWEST ADDR  ***SMT
        BHI    R01.1
        MOV    #STKFLO,-(6)                ; SET ERROR NUMBER
        JSR    X1,RRGEL                    ; GO TO ERROR ROUTINE
R01.1:  SUB    (2)+,X6                     ; ADJUST X6
        MOV    X1,-(6)                     ; DUMP LINK
        MOV    X5,-(6)                     ; DUMP OLD X5
        MOV    X6,X5                       ; SET NEW X5=X6
        JMP    (2)                         ; RETURN TO CODING
;
; PROCEDURE EXIT - NO RESULT
;
R02:
R02.0:  TST    (6)+                        ; DISCARD OLD X2
        MOV    (6)+,X5                     ; RESET X5
        MOV    (6)+,X2                     ; PICK UP LINK
        MOV    X5,X6                       ; RESET X6
        JMP    (2)                         ; RETURN
;
; PROCEDURE EXIT - ONE WORD RESULT
;
R03:
R03.0:  TST    (6)+                        ; DISCARD OLD X2
        MOV    (6)+,X1                     ; PICK UP RESULT FROM STACK
        MOV    (6)+,X5                     ; RESET X5
        MOV    (6)+,X4                     ; PICK UP LINK
        ADD    (2),X6                      ; RESET X6
R03.1:  MOV    X1,-(6)                     ; DUMP RESULT
        JMP    (4)                         ; RETURN
;
; PROCEDURE EXIT - DOUBLE WORD RESULT
;
R04:
R04.0:  TST    (6)+                        ; DISCARD OLD X2
        MOV    (6)+,X1                     ; FIRST WORD OF RESULT FROM STACK
        MOV    (6)+,X3                     ; 2ND.  WORD OF RESULT FROM STACK
        MOV    (6)+,X5                     ; RESET X5
        MOV    (6)+,X4                     ; PICK UP LINK
        ADD    (2),X6                      ; RESET X6
        MOV    X3,-(6)                     ; DUMP RESULT, FIRST WORD
        BR     R03.1
;
; GOTO GENERAL LABEL - USES STACK UNWIND ROUTINE R25
;
R09:
R09.0:  TST    (6)+                        ; IGNORE LINK
        JSR    X2,R25.0                    ; UNWIND, GOES TO LABEL IF OK
        MOV    #LABERR,-(6)                ; SET ERROR NUMBER
        JSR    X1,RRGEL                    ; GO TO ERROR ROUTINE
;
; STACK UNWIND ROUTINE
; GOES DIRECTLY TO LABEL IF FOUND
; RETURNS WITH STACK UNCHANGED IF NOT FOUND
;
R25:
R25.0:  TST    (6)+                        ; IGNORE OLD X2
        MOV    (6)+,X1                     ; LABEL X7
```

```
        MOV     (6)+,X4                 ; LABEL X5
R25.1:  CMP     X4,X5                   ; FOUND IT?
        BEQ     R25.2
        CMP     X5,(5)                  ; END OF STACK ?
        BEQ     R25.3
        MOV     (5),X5                  ; UNWIND ONE LEVEL
        BR      R25.1
R25.2:  MOV     X5,X6                   ; RESET X6
        JMP     (1)                     ; GOTO LABEL
R25.3:  MOV     X6,X5                   ; RESET STACK POINTER
        JMP     (2)                     ; RETURN TO CALLER
       .PAGE
       .SBTTL FIXED POINT MUL, BASIC & EIS
;
; ALL FIXED POINT OVERFLOW DETECTION LOGIC IS COMMENTED OUT.
; SUCH ERRORS WOULD INVOLVE A BR (OR JMP) TO A LABEL OFLO, WHICH IS
; CURRENTLY NOT DEFINED.
; USERS MAY DEFINE OFLO IF THEY WISH.
;
.IF NDF ZEISIL
;
; R05   MULTIPLY, SINGLE LENGTH RESULT
; SETS FLAG AND USES R06
; EXEC TIME, FASTEST=58.5, SLOWEST=455.2
;
;
; R06   MULTIPLY, DOUBLE LENGTH RESULT
; DESTROYS REGISTERS X1 AND X2
; TESTS FLAG AT END OF ROUTINE TO DETERMINE
; IF IS SINGLE LENGTH RESULT (R05 CALL)
; EXEC TIME, FASTEST=54.6, SLOWEST=451.3
; STACK USAGE - 3+4 WORDS
;
;
R05:
R05.0:  CLR     X1                      ; SET EXIT TYPE FLAG
        BR      R06.1
;
;
R06:
R06.0:  MOV     #1,X1                   ; SET EXIT TYPE FLAG
R06.1:  MOV      X2,(6)
        MOV X3,-(6)                     ; DUMP X3
.IF  NDF  ZHWMD
        MOV X4,-(6)                     ; DUMP X4
        SWAB  X1                        ; PUT EXIT TYPE INTO TOP BYTE
        MOV   6(6),X2                   ; PICK UP FIRST OPERAND
        BGE   R06.2
        NEG   X2                        ; NEGATE IF NEGATIVE
        INCB  X1                        ; REMEMBER SIGN
R06.2:  MOV   8.(6),X3                  ; PICK UP 2ND OPERAND
        BGE   R06.3
        NEG   X3                        ; NEGATE IF NEGATIVE
        DECB  X1                        ; REMEMBER SIGN
R06.3:  CMP   X2,X3                     ; X2:=SMALLER OF X2,X3
        BLE   R06.4
        MOV   X2,X4                     ; SWAP X2 AND X3
        MOV   X3,X2
        MOV   X4,X3
R06.4:  CLR   X4                        ; X4.X3 IS MULTIPLICAND
        MOV   X1,-(6)                   ; DUMP X1, SIGN FLAG
        MOV   X2,-(6)                   ; DUMP X2, MULTIPLIER
```

```
        CLR   X2                        ; PRODUCT L.S. HALF
        CLR   X1                        ; PRODUCT M.S. HALF
R06.5:  TST   (6)                       ; MULTIPLIER EXHAUSTED YET?
        BEQ   R06.7
        ROR   (6)                       ; TEST L.S. BIT
        BCC   R06.6
        ADD   X3,X2                     ; 2 WORD ADD
        ADC   X1
        ADD   X4,X1
R06.6:  ASL   X3                        ; DOUBLE MULTIPLICAND
        ROL   X4
        BR    R06.5
R06.7:  TST   (6)+                      ; SKIP OLD MULTIPLIER
        TSTB  (6)                       ; TEST SIGN BIT
        BEQ   R06.8
        NEG   X1                        ; 2 WORD NEGATE
        NEG   X2
        SBC   X1
R06.8:  MOV   X2,10.(6)                 ; DUMP RESULT L.S. HALF
        MOV   X1,8.(6)                  ; DITTO M.S. HALF
        MOV   (6)+,X1                   ; ACCESS TYPE FLAG
        MOV   (6)+,X4                   ; RESTORE X3 AND X4
        MOV   (6)+,X3
        MOV   (6)+,X2                   ; PICK UP LINK
        SWAB  X1                        ; ACCESS EXIT TYPE FLAG
.IFF                    ;(ZHWMD)
        MOV 6(6),X2                     ; FIRST OPERAND
       CLR     X3
        MUL 4.(6),X2                    ; MULTIPLY BY SECOND
        MOV X3,6(6)                     ; RESULT L.S. HALF
        MOV X2,4(6)                     ; RESULT M.S. HALF
        MOV (6)+,X3                     ; RESTORE X3
       MOV    (6)+,X2
        TST X1                           ; RESULT TYPE FLAG
.IFTF                   ;(ZHWMD)
        BNE   R06.9
        TST   (6)+                       ; REJECT M.S. HALF
R06.9:  JMP   (2)                       ; RETURN
        .PAGE
        .SBTTL FIXED POINT DIV, BASIC & EIS
;
;
; R07    DIVIDE, SINGLE LENGTH DIVIDEND
; EXTENDS DIVIDEND AND USES R08
; EXEC TIME, ADDS 23.3 TO R08 IF DIVIDEND +VE, 27.3 IF -VE
;
;
; R08   DIVIDE, DOUBLE LENGTH DIVIDEND
; DIVISOR AND DIVIDEND ON STACK
; RESULT ON STACK, REMAINDER IN X1
; DESTROYS REGISTERS X1 AND X2
; EXEC TIME, WORST CASE=390
; STACK USAGE - 4+4 WORDS
;
;
R07:
R07.0:  MOV   X6,X1                     ; COPY X6
        MOV   (1)+,-(6)                 ; COPY OLD X2
        MOV   (1),-2(1)                 ; COPY DIVISOR
        CLR   (1)+                      ; CLEAR M.S. HALF OF DIVIDEND
        TST   (1)                       ; TEST L.S. HALF OF DIVIDEND
        BGE   R08.0
```

0-8-7

```
        DEC     -2(1)                           ; EXTEND SIGN BIT IF NEGATIVE
;
;
R08:
R08.0:  MOV     X2,(6)                          ; REMEMBER LINK
        MOV      X3,-(6)                        ;DUMP X3 AND X4
        MOV      X4,-(6)
.IFT                    ;(ZHWMD)
        CLR     X3                              ; SIGN FLAG
        MOV     6(6),X2                         ; PICK UP DIVISOR
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       BEQ     OFLO                            ; OVERFLOW IF NEGATIVE
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        BGT     R08.1
        NEG     X2                              ; NEGATE IF NEGATIVE
        INC     X3                              ; REMEMBER SIGN
R08.1:  MOV     8.(6),X4                        ; DIVIDEND M.S. HALF
        MOV     10.(6),X1                       ; DIVIDEND L.S. HALF
        MOV     X4,-(6)                         ; REMEMBER SIGN
        BGE     R08.2
        NEG     X4                              ; NEGATE IF NEGATIVE
        NEG     X1
        SBC     X4
        DEC     X3                              ; SET SIGN FLAG
R08.2:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       CMP     X1,X2                           ; OVERFLOW IF M.S. HALF GREATER
;       BHI     OFLO                            ; THAN DIVISOR (UNSIGNED TEST)
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        MOV     X3,-(6)                         ; DUMP SIGN FLAG
        MOV     #16.,X3                         ; SET LOOP COUNT
R08.3:  ASL     X1                              ; 2 WORD SHIFT
        ROL     X4
        CMP     X4,X2
        BLO     R08.4
        SUB     X2,X4                           ; SUB. DIVISOR IF M.S. HALF
                                                ; EXCEEDS IT
        INC     X1                              ; PUT BIT INTO RESULT
R08.4:  DEC     X3                              ; LOOP IF MORE TO DO
        BGT     R08.3
        MOV     (6)+,X3                         ; RESTORE SIGN FLAG
        TST     (6)+                            ; CORRECT SIGN OF REMAINDER
        BGE     R08.5
        NEG     X4
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       NEG     X2                              ; CHANGE SIGN OF DIVISOR
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R08.5:  TST     X3                              ; CORRECT SIGN OF QUOTIENT
        BEQ     R08.6
        NEG     X1
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       CMP     X0,#100000                      ; QUOTIENT TOO LARGE?
;       BEQ     OFLO
;       BR      R08.7
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R08.6:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       CMP     (6),X2                          ; DIVISOR=ORIG. M.S. HALF?
;       BEQ     OFLO
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R08.7:  MOV     X4,8.(6)                        ; DUMP REMAINDER
        MOV     (6)+,X4                         ; RESTORE X3 AND X4
```

```
        MOV     (6)+,X3
        MOV     (6)+,X2                         ; PICK UP LINK
        TST     (6)+                            ; SKIP OLD DIVISOR
        MOV     X1,2(6)                         ; DUMP RESULT
        MOV     (6)+,X1                         ; PICK UP REMAINDER
.IFF                    ;(ZHWMD)
        CLR     X1                              ;SIGN FLAG
        MOV     6(6),X4                         ;DIVISOR
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       BEQ     OFLO
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        BGT     R08.1                           ; BRANCH IF POSITIVE
        NEG     X4                              ;NEGATE IF NEGATIVE
        INC     X1                              ;REMEMBER SIGN
R08.1:  MOV     8.(6),X2                        ; DIVIDEND M.S. HALF
        MOV     10.(6),X3                       ; DIVIDEND L.S. HALF
        MOV     X2,-(6)                         ; REMEMBER SIGN
        BGE     R08.2                           ; BRANCH IF POSITIVE
        NEG     X2
        NEG     X3                              ;DOUBLE WORD NEGATE
        SBC     X2                              ;
        DEC     X1                              ;REMEMBER SIGN
R08.2:
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
;       TST     X4                              ; TEST FOR DIV.BY ZERO
;       BEQ     OFLO
;       CMP     X3,X4                           ; OVERFLOW IF M.S.>DIVISOR
;       BHI     OFLO
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        DIV     X4,X2                           ; DO DIVISION
        BCC     R08.5                           ; TEST FOR DIV.BY ZERO
R08.3:  MOV     #77777,X2                       ; MAXIMUM RESULT
        CLR     X3                              ; AND ZERO REMAINDER
        BR      R08.6
R08.5:  BVS     R08.3                           ; OVERFLOW
R08.6:  TST     (6)+                            ; TEST SIGN OF DIVIDEND
        BGE     R08.7
        NEG     X3
R08.7: TST      X1                              ; SIGN FLAG
       BEQ      R08.8
       NEG      X2                              ;NEGATE QUOTIENT IF NEC.
R08.8:  MOV     X3,X1                           ;REMAINDER IN X1 ON EXIT
        MOV     X2,10.(6)                       ;QUOTIENT ON STACK
        MOV     (6)+,X4                         ;RESTORE X4
        MOV     (6)+,X3                         ;RESTORE X3
        MOV     (6)+,X2                         ;RESTORE X2
        CMP     (6)+,(6)+                       ;DISCARD INPUT PARAMS.
.IFTF                   ;(ZHWMD)
        JMP     (2)                             ;RETURN
;
       .PAGE
       .SBTTL ARITHMETIC SHIFTS, BASIC
;
.IFT                    ;(ZHWMD)
; R10   SRA   ARITHMETIC RIGHT SHIFT
; DOUBLE LENGTH OPERAND AND RESULT
; DESTROYS REGISTERS X1 AND X2
; THIS VERSION EXCHANGES BYTES WHERE POSSIBLE
; EXEC TIME=35.1+9.5*N IF N<8, + ANOTHER 26.9 PER 8 BITS THEREAFTER
;
;
; R11   SLA   ARITHMETIC LEFT SHIFT
```

0-8-8

```
; DOUBLE LENGTH OPERAND AND RESULT                                    (0-8-9)
; DESTROYS REGISTERS X1 AND X2
; THIS VERSION EXCHANGES BYTES WHERE POSSIBLE
; EXEC TIME=35.1+9.5*N IF N<8, + ANOTHER 24.6 PER 8 BITS THEREAFTER
;
;
; R12   SIGNED ARITHMETIC SHIFT
; SINGLE LENGTH OPERAND AND RESULT
; EXTENDS OPERAND, SETS SPECIAL RETURN ADDRESS, THEN USES R13
; EXEC TIME, ADDS 34.0 TO R13 IF OPERAND +VE, 38.0 IF -VE
;
;
; R13   SIGNED ARITHMETIC SHIFT
; DOUBLE LENGTH OPERAND AND RESULT
; TESTS SIGN OF SHIFT COUNT AND USES R10 OR R11
; EXEC TIME, ADDS 14.2 TO R10 AND 10.2 TO R11
;
; STACK USAGE FOR ALL THE ABOVE - 4+2 WORDS
;
R10:
R10.0:    MOV    X2,(6)                  ; DUMP LINK ADDRESS
          MOV    X3,-(6)                 ; DUMP X3
          MOV    4(6),X3                 ; SHIFT COUNT
R10.13:   MOV    6(6),X1                 ; OPERAND M.S. HALF
          MOV    8.(6),X2                ; OPERAND L.S. HALF
R10.1:    CMPB   X3,#8.                  ; BYTE SHIFT?
          BLO    R10.3                   ; GOTO BIT SHIFT
; BYTE SHIFT
          SWAB   X2                      ; SHIFT LOWEST BYTE
          MOVB   X2,-(6)                 ; EXTRACT LOWEST BYTE OF RESULT
          MOVB   X1,1(6)                 ; AND NEXT BYTE UP
          MOV    (6)+,X2                 ; PUT BACK INTO X2
          SWAB   X1                      ; MOVE TOP BYTE DOWN
          MOVB   X1,X1                   ; EXTEND SIGN BIT
          SUB    #8.,X3                  ; DECREMENT COUNT
          BR     R10.1                   ; TRY AGAIN
; BIT SHIFT
R10.2:    ASR    X1                      ; 2 WORD SHIFT
          ROR    X2
R10.3:    DECB   X3                      ; DECREMENT COUNT
          BGE    R10.2                   ; MORE TO DO?
          BR     R11.4                   ; COMMON EXIT ROUTINE
;
;
R11:
R11.0:    MOV    X2,(6)                  ; DUMP LINK ADDRESS
          MOV    X3,-(6)                 ; DUMP X3
          MOV    4(6),X3                 ; SHIFT COUNT
R11.13:   MOV    6(6),X1                 ; OPERAND M.S. HALF
          MOV    8.(6),X2                ; OPERAND L.S. HALF
R11.1:    CMPB   X3,#8.                  ; BYTE SHIFT?
          BLO    R11.3                   ; GOTO BIT SHIFT
; BYTE SHIFT
          SWAB   X2                      ; SHIFT UP BOTTOM BYTE
          MOVB   X2,-(6)                 ; BOTTOM BYTE FOR NEW X1
          MOVB   X1,1(6)                 ; TOP DITTO
          MOV    (6)+,X1                 ; COPY TO X1
          CLRB   X2                      ; ZEROS IN L.S. BYTE
          SUB    #8.,X3                  ; DECREMENT COUNT
          BR     R11.1                   ; TRY AGAIN
; BIT SHIFT
R11.2:    ASL    X2                      ; 2 WORD SHIFT
```

```
        ROL     X1
R11.3:  DECB    X3                      ; DECREMENT COUNT
        BGE     R11.2                   ; MORE TO DO?
R11.4:  MOV     (6)+,X3                 ; RESTORE X3
        MOV     X2,6(6)                 ; DUMP RESULT L.S. HALF
        MOV     (6)+,X2                 ; PICK UP LINK
        TST     (6)+                    ; IGNORE OLD SHIFT COUNT
        MOV     X1,(6)                  ; DUMP RESULT M.S. HALF
        JMP     (2)                     ; RETURN
;
;
R12:
R12.0:  MOV     X6,X1                   ; COPY X6
        MOV     2(1),-(6)               ; COPY SHIFT COUNT
        CLR     (1)+                    ; CLEAR OPERAND M.S. HALF
        MOV     2(1),(1)                ; COPY OPERAND L.S. HALF
        BGE     R12.1
        DEC     -2(1)                   ; EXTEND SIGN BIT
R12.1:  CMP     -(6),(1)+               ; ADJUST POINTERS
        MOV     X2,(1)                  ; DUMP OLD LINK
        MOV     #R12.2,X2               ; SET SPECIAL RETURN ADDRESS
;
;
R13:
R13.0:  MOV     X2,(6)                  ; REMEMBER LINK ADDRESS
        MOV     X3,-(6)                 ; DUMP X3
        MOV     4(6),X3                 ; PICK UP SIGNED SHIFT COUNT
        BGE     R11.13                  ; SHIFT LEFT IF POSITIVE
        NEG     X3                      ; NEGATE IF NEGATIVE
        BR      R10.13                  ; SHIFT RIGHT
;
;
R12.2:  TST     (6)+                    ; IGNORE M.S. HALF OF RESULT
        MOV     (6)+,X1                 ; PICK UP L.S. HALF
        MOV     (6)+,X2                 ; PICK UP LINK
        MOV     X1,-(6)                 ; DUMP RESULT
        JMP     (2)                     ; RETURN
      .PAGE
      .SBTTL ARITHMETIC SHIFTS, EIS
.IFF                    ;(ZHWMD)
; R10 SRA ARITHMETIC RIGHT SHIFT.
; DOUBLE LENGTH OPERAND AND RESULT
; DESTROYS X2
;
; R11 SLA ARITHMETIC LEFT SHIFT
; DOUBLE LENGTH OPERAND AND RESULT
; DESTROYS X2
;
; R12  SIGNED ARITHMETIC SHIFT
; SINGLE LENGTH OPERAND AND RESULT
; DESTROYS  X2
;
; R13  SIGNED ARITHMETIC SHIFT
; DOUBLE LENGTH OPERAND AND RESULT
; DESTROYS X2
;
R10:
R10.0:  NEG     2(6)                    ;RIGHT SHIFT IS -VE IN ASH,ASHC
R11:
R11.0:
R13:
R13.0:  MOV     X2,(6)                  ;SAVE LINK
```

```
          MOV     X3,-(6)                 ;SAVE X3
          MOV     6(6),X2                 ;MS HALF OPERAND
          MOV     8.(6),X3                ;LS HALF OPERAND
          ASHC    4(6),X2                 ;DO SHIFT
          MOV     X2,6(6)                 ;MS HALF RESULT
          MOV     X3,8.(6)                ;LS HALF RESULT
          MOV     (6)+,X3                 ;RESTORE X3
          MOV     (6)+,X2                 ;RESTORE LINK
          TST     (6)+                    ;OLD SHIFT COUNT
          JMP     (2)                     ;RETURN
;
R12:
R12.0:
          MOV     X3,-(6)                 ;SAVE X3
          MOV     6(6),X3                 ;OPERAND
          ASH     4(6),X3                 ;DO SHIFT
          MOV     X3,6(6)                 ;RESULT ON STACK
          MOV     (6)+,X3                 ;RESTORE X3
          ADD      #4,X6                   ;SET STACK POINTER FOR EXIT
          JMP     (2)                     ;RETURN
;
;
.ENDC                       ;(ZHWMD)
.ENDC                       ;(ZEISIL)
        .PAGE
        .SBTTL LOGICAL SHIFTS, BASIC
.IF NDF ZHWMD
; R14   SRL   SHIFT RIGHT LOGICAL
; SINGLE LENGTH OPERAND AND RESULT
; DESTROYS REGISTERS X1 AND X2
; THIS VERSION EXCHANGES BYTES WHERE POSSIBLE
; CALLING SEQUENCE IS   JSR   X2,R14
; SHIFT COUNT AND OPERAND ARE TOP TWO WORDS ON STACK
; RESULT LEFT ON TOP OF STACK
; EXEC TIME=20.1+10.1*N IF N<8, OTHERWISE 38.9+10.1*(N-8)
;
;
; R15   SLL   SHIFT LEFT LOGICAL
; SIMILAR SPEC. TO R14
; EXEC TIME=20.1+8.6*N IF N<8, OTHERWISE 38.9+8.6*(N-8)
;
;
; R16   SHL   SIGNED LOGICAL SHIFT
; TESTS SIGN OF SHIFT COUNT AND THEN USES R14 OR R15
; EXEC TIME, ADDS 14.2 TO R14 AND 10.2 TO R15
;
; STACK USAGE FOR ALL THE ABOVE - 3+0 WORDS
;
R14:
R14.0:  TST   (6)+                      ; IGNORE OLD X2
        MOV   (6)+,X1                   ; PICK UP SHIFT COUNT
R14.1:  CMPB  X1,#8.                    ; BYTE SHIFT?
        BLO   R14.3                     ; GOTO BIT SHIFT
; BYTE SHIFT
        CLRB  (6)                       ; ZEROS (FOR TOP BYTE)
        SWAB  (6)
        SUB   #8.,X1                    ; DECREMENT COUNT
        BR    R14.1
; BIT SHIFT
R14.2:  CLC                             ; LOGICAL RIGHT SHIFT
        ROR   (6)
R14.3:  DECB  X1                        ; DECREMENT COUNT
```

```
        BGE     R14.2                   ; MORE TO DO?
        JMP     (2)                     ; RETURN
;
;
R15:
R15.0:  TST     (6)+                    ; IGNORE OLD X2
        MOV     (6)+,X1                 ; PICK UP SHIFT COUNT
R15.1:  CMPB    X1,#8.                  ; BYTE SHIFT?
        BLO     R15.3                   ; GOTO BIT SHIFT
; BYTE SHIFT
        SWAB    (6)                     ; SHIFT BOTTOM BYTE
        CLRB    (6)                     ; ZEROS IN L.S. BYTE
        SUB     #8.,X1                  ; DECREMENT COUNT
        BR      R15.1                   ; TRY AGAIN
; BIT SHIFT
R15.2:  ASL     (6)                     ; SHIFT LEFT
R15.3:  DECB    X1                      ; DECREMENT COUNT
        BGE     R15.2                   ; MORE TO DO?
        JMP     (2)                     ; RETURN
;
;
R16:
R16.0:  TST     (6)+                    ; IGNORE OLD X2
        MOV     (6)+,X1                 ; PICK UP SHIFT COUNT
        BGE     R15.1                   ; >=0, GOTO LEFT SHIFT
        NEG     X1                      ; NEGATE IF NEGATIVE
        BR      R14.1                   ; RIGHT SHIFT
;
        .PAGE
        .SBTTL LOGICAL SHIFTS, EIS
.IFF                    ;(ZHWMD)
; R14   SRL    SHIFT RIGHT LOGICAL
; SINGLE LENGTH OPERAND AND RESULT
; DESTROYS X1,X2
;
; R15   SLL    SHIFT LEFT LOGICAL
; SIMILAR TO R14
;
; R16   SHL    SIGNED LOGICAL SHIFT
; TESTS SIGN OF SHIFT COUNT AND USES R14 OR R15
;
;
;
R14:
R14.0:  NEG     2(6)                    ;NEGATE FOR ASHC
;
R15.0:
;
R15:
;
R16:
;
R16.0:
;
        MOV     X2,X1                   ;LINK VACATING X2
        CLR     X2                      ;PROVIDES LEADING ZEROS IF RIGHT SHIFT
        MOV     2(6),(6)                 ;SHIFT COUNT
       MOV      X3,2(6)                ;SAVE X3
       MOV      4(6),X3                ;GET OPERAND
        ASHC    (6)+,X2                  ;DO SHIFT AND DISCARD COUNT
        MOV     X3,2(6)                  ;RESULT REPLACES OPERAND ON STACK
       MOV      (6)+,X3                ;RESTORE X3
```

```
        JMP     (1)
;
.ENDC                       ;(ZHWMD)
        .PAGE
        .SBTTL ARRAY BOUND & ACCESS, COMMON

;
;
; R17 - R22     ARRAY ALIGNMENT
;               BOUNDS CHECKING
;               ADDRESS CALCULATION
;
; FOR ARRAYS OF
;       BYTES (R17)    EXEC TIME=15.2
;       INT,FRAC,PROC,STACK (R18)    EXEC TIME=20.1
;       LABELS (R19)    EXEC TIME=25.0
;       REALS (R20)    EXEC TIME=25.0
;       RECORDS (R21)    TYPICAL EXEC TIME=130.0
;
; MODIFIER IN X3, ARRAY BASE IN X4
; X3 NOT DESTROYED, FINAL ADDRESS RETURNED IN X4
;
; R22 ALIGNS THE MODIFIER FOR ARRAYS OF RECORDS, RESULT RETURNED IN X3
; TYPICAL EXEC TIME = 120.0
;
; ALL THESE ROUTINES DESTROY X1
; R21 AND R22 ALSO DESTROY X2 AND USE 3 WORDS OF STACK
;
; CALLING SEQUENCE    JSR    X2,RNN
; AND FOR R21 AND R22    JSR X2,RNN
;                        .WORD 'MULTIPLICATION FACTOR'
;
;
R17:
R17.0:  MOV     #-1,(6)                 ; BYTE ALIGNMENT FACTOR
        MOV     X3,X1                   ; BYTE ENTRY
R17.1:  TST     X1                      ; ERROR IF <=0
        BLE     R17.2
        ADD     (6),X4                  ; ALIGN TO LENGTH WORD
        CMP     X1,(4)                  ; CHECK WITH ARRAY LENGTH
        BGT     R17.2
        ADD     X1,X4                   ; CALCULATE FINAL ADDRESS
        SUB     (6)+,X4                 ; SUBTRACT ALIGNMENT
        JMP     (2)                     ; RETURN
R17.2:  MOV     #ARRERR,-(6)            ; SET ERROR NUMBER
        JSR     X1,RRGEL                 ; GO TO ERROR ROUTINE
;
;
R18:
R18.0:  MOV     X3,X1                   ; SINGLE WORD ENTRY
        CLR     (6)                     ; WORD ALIGNMENT FACTOR
R18.1:  ASL     X1                      ; ALIGN
        BR      R17.1
;
;
R19:
R19.0:                                  ; DOUBLE WORD ENTRY
R20:
R20.0:                                  ; REALS ENTRY
        MOV     #2,(6)                  ; DOUBLE WORD ALIGNMENT FACTOR
        MOV     X3,X1
        ASL     X1                      ; ALIGN
```

```
        BR      R18.1
;
.IF  NDF  ZHWMD
        .SBTTL ARRAY ACCESS, BASIC
;
; BASIC VERSION OF R21 & R22
;;
R21:
R21.0:  MOV     (2),(6)                 ; RECORD LENGTH
        SUB     #2,(6)                  ; ALIGNMENT FACTOR
        CLR     -(6)                    ; RECORD ENTRY, SET FLAG
        BR      R22.1
;
;
R22:
R22.0:  MOV     #1,-(6)                 ; RECORD MODIFIER ENTRY, SET FLAG
R22.1:  MOV     (2)+,X1                 ; PICK UP RECORD SIZE
        MOV     X2,-(6)                 ; INCREMENT LINK ADDRESS AND DUMP
        MOV     X3,X2                   ; COPY X3
        CLR     -(6)                    ; CLEAR RESULT FOR SIMPLE MULT
R22.2:  TST X1                          ; MORE TO DO?
        BEQ     R22.4
        ROR     X1                      ; EXAMINE L.S. BIT
        BCC     R22.3
        ADD     X2,(6)                  ; ADD INTO RESULT
R22.3:  ASL     X2                      ; SCALE UP BY 2
        BR      R22.2
R22.4:  MOV     (6)+,X1                 ; PICK UP RESULT
        MOV     (6)+,X2                 ; PICK UP LINK ADDRESS
        TST     (6)+                    ; TEST FLAG
        BEQ     R17.1
        MOV     X1,X3                   ; SET RESULT OF R22
        TST     (6)+                    ; RESET X6
        JMP     (2)                     ; RETURN
;
.IFF                            ;(ZHWMD)
        .SBTTL ARRAY ACCESS, EIS
;
; EIS VERSION OF R21 & R22
;
; ON ENTRY: %3 CONTAINS SUBSCRIPT, JSR %2,R21 ,.WORD RECORD LENGTH
; ON EXIT : %3 UNCHANGED, %4 POINTS TO REQUIRED RECORD
;
R21:
R21.0:  MOV      (2),(6)            ;RECORD LENGTH
        SUB      #2,(6)             ; ALIGNMENT FACTOR
        MOV      (2)+,X1            ; GET RECORD SIZE, INCREMENT LINK
        MUL      X3,X1              ;DISPLACEMENT IN X1
        BR       R17.1              ; FOR BOUND CHECK
;
; ON ENTRY: %3 CONTAINS SUBSCRIPT, JSR %2,R22, .WORD RECORD LENGTH
; ON EXIT : %3 CONTAINS SUBSCRIPT * RECORD LENGTH
;
R22:
R22.0:  MUL      (2)+,X3            ; DO MUL & UPDATE LINK ADDRESS
        RTS      X2                 ; RETURN
;
;
.ENDC                   ;(ZHWMD)
;
        .PAGE
.IF  DF  ZFLPT
```

```
;
;
; RTL/2 PDP-11 FLOATING POINT CONTROL ROUTINES
;
;
.IF NDF Z45FP
;
; USES TWO WORD REAL FORMAT
; ROUTINES DO NOT USE X0, AND PRESERVE X3,X4 AND X5
; TIMES GIVEN ARE IN MICRO-SECONDS
; TOTAL SIZE = 409 WORDS
;
;
;
.IF NDF ZFPIL
        .SBTTL FLOATING COMMON ENTRY, BASIC
;
; COMMON ENTRY ROUTINE FOR FLOATING POINT ADD,
; SUBTRACT, MULTIPLY AND DIVIDE
; EXPECTS TO FIND TWO REALS DUMPED ON THE STACK
; RETURNS WITH THE EXPONENTS IN X1 AND X4,
; THE SECOND SIGN IN THE CARRY,
; NEV OF THE SIGN BITS ON TOP OF THE X6 STACK,
; AND LEAVES THE FRACTIONS OF THE REALS WITH THE TOP BYTE CLEAR AND
; THE 'NORMALISE' BIT INSERTED
; X2 POINTS TO THE M.S. HALF OF THE SECOND (HIGHEST ADDRESSED) REAL
;
FPENT:   MOV    X2,2(6)                  ;REMEMBER LINK
         MOV    X4,-(6)
         MOV    X6,X2             ;POINT X2 AT FIRST REAL
         ADD    #6.,X2
         MOV    (2)+,X1                       ; PICK TOP OF REAL
         CLR    -(6)                        ;CLEAR FOR SIGN
         ROL    X1                      ; SHIFT OUT SIGN OF FIRST
         ROR     (6)               ; REMEMBER FIRST SIGN
         CLRB   X1                        ; CLEAR BOTTOM BYTE
         SWAB   X1                        ; PUT EXPONENT IN BOTTOM BYTE
         CLRB   -(2)                      ; CLEAR TOP BYTE OF REAL
         BISB   #200,-(2)                  ;PUT IN 'NORMALISE' BIT
         CMP    (2)+,(2)+                 ; POINT TO SECOND REAL
         MOV    (2)+,X4                   ; REPEAT FOR SECOND REAL
         ADD    X4,(6)                     ; NEV SIGN BITS
         ASL    X4                         ; SHIFT OUT SIGN
         ROR    (6)                        ; STORE SECOND SIGN
         CLRB   X4
         SWAB   X4
         CLRB   -(2)
         BISB    #200,-(2)                  ;
        ROL      (6)                        ; SECOND SIGN TO CARRY
         JMP    (3)                       ; RETURN
;
        .PAGE
.IFTF                   ;(ZFPIL)
        .SBTTL FLOATING COMMON EXIT, BASIC
;
; GENERAL PURPOSE ROUND NORMALISE AND EXIT,USED BY ALL THE
; FP CONTROL ROUTINES YIELDING A REAL RESULT - ENTERED WITH
; EXPONENT IN X1,SIGN ON STACK,M.S. PART OF MANTISSA IN X4,
; L.S. PART OF MANTISSA IN X3.
;
;
;
```

0-8-16

```
RNDTST:
        BIT     #177400,X4              ; TEST IF READY TO ROUND
        BNE     ALIGN                   ;I.E NEARLY ALIGNED.NOT
                                        ;SHIFTED DOWN TO ROUND POINT
        ADC     X3                      ; ROUND
        ADC     X4
ALIGN:
        ASR     X4                      ;ENTRY POINT
        ROR     X3
        INC     X1
        BIT     #177600,X4              ;TEST ALIGNMENT CORRECT
        BNE     RNDTST                  ;WHEN SHIFTED UP.LS BIT IN CARRY
RUNUP:
        ROL     X3
        ROL     X4
        DEC     X1
        TSTB    X4                      ;IS MANTISSA ALIGNED AT
                                        ;BYTE BOUNDRY
        BMI     FLOTST                  ;JUMP IF ALIGNED,NOT ALIGNED
        BNE     RUNUP                   ;SHIFT UP MORE
        TST     X3                      ;CONVENTIONAL ZERO
        BNE     RUNUP                   ;GIVEN TIME IT WILL ALIGN
UFLO:                                   ;FLOATING POINT UNDERFLOW ENTRY
        CLR     X3                      ;UNDERFLOW.RETURN
        CLR     X4                      ;CONVENTIONAL ZERO
        TST     (6)+                    ;DITCH SIGN
        BR      FPEXIT
FLOTST:
        TST     X1                      ;UFLO IF EXPONENT ZERO OR NEG
        BLE     UFLO
        SWAB    X1
        BEQ     NORM                    ;BRANCH IF NOT OFLO
OVFLO:                                  ;FLOATING POINT OVERFLOW ENTRY
        MOV     #377,X4                 ;BIG REAL FRAC PART
        MOVB    X4,X3
        MOV     #177400,X1
NORM:
        ROL     (6)+                    ;FETCH SIGN TO CARRY
        ROR     X1                      ;SHIFT EXP ADD SIGN BIT
        BIC     #200,X4                 ;CLEAR NORMALISE BIT
        BIS     X1,X4                   ;COMBINE EXP AND MANTISSA
FPEXIT:
        MOV     X3,12.(6)               ;DUMP L.S. HALF OF RESULT
         MOV    X4,10.(6)               ; DUMP M.S. HALF OF RESULT
         MOV    (6)+,X4
         MOV    (6)+,X3
         MOV    (6)+,X2                 ; LINK ADDRESS
         CMP    (6)+,(6)+               ; SKIP OLD OPERAND
         JMP    (2)                     ; RETURN
;
;
        .PAGE
.IFT                    ;(ZFPIL)
;
        .SBTTL FLOATING ADD & SUB, BASIC
;
; FLOATING POINT SUBTRACT
; CHANGES SIGN OF SECOND OPERAND AND USES ADD
;
R27:
R27.0:   ADD    #100000,2(6)
;
```

```
;
;
;
;  FLOATING POINT ADD
;
R26:
R26.0:
         JSR    X3,FPENT                ; USE COMMON ENTRY ROUTINE
         BCC    R26.2                    ;SECOND REAL IS +VE
R26.1:
         NEG    2(2)                     ;DOUBLE LENGTH NEGATE
         ADC    (2)
         NEG    (2)
         COM    (6)                      ;RECOVER SIGN FROM NEV
R26.2:
         CMP    -(2),-(2)                ;POINT TO FIRST REAL
         TST    (6)
         BMI    R26.1                    ;FIRST REAL IS NEGATIVE
         MOV    X4,(6)                   ;DUMP SECOND EXP. OVER SIGN
         MOV    X6,X2                    ;POINT TO FIRST REAL M.S.
         ADD    #8.,X2
         MOV    (2)+,X4                  ;FIRST MANTISSA TO X3,X4
         MOV    (2)+,X3                  ;X2 POINTS TO SECOND
         SUB    X1,(6)                   ;SKIP SWAB IF FIRST EXPONENT
         BLE    R26.8                    ;IS LARGER
         MOV    (2)+,X4                  ;SECOND(LARGER) MANTISSA
         MOV    (2),X3                   ;TO X3,X4
         SUB    #6.,X2                   ;X2 POINTS TO FIRST(SMALLER)
         ADD    (6),X1                   ;X1 BECOMES LARGER EXP.
         BR     R26.3
R26.8:
         NEG    (6)                      ;SHIFT COUNT = ABS SCALE DIFF
R26.3:
         CMP    (6),#24.
         BGE    R26.7                    ;COMPARATIVELY INSIGNIFICANT
         CMP    (6),X1                   ;OR CONVENTIONAL ZERO
         BEQ    R26.7
         TST    (6)
         BEQ    R26.6                    ;BOTH SCALINGS SAME
         ASL    X3                       ;SHIFT LARGER UP 1 TO ROUND
         ROL    X4                       ;SCOPE FOR ROUNDING IN
         DEC    X1                       ;COMMON NORMALISE ROUTINE
         BR     R26.5
R26.4:
         ASR    (2)                      ;SHIFT SMALLER DOWN TO
         ROR    2(2)                     ;EQUATE SCALES
R26.5:
         DEC    (6)
         BNE    R26.4
R26.6:
         ADD    2(2),X3                  ;DOUBLE LENGTH ADD
         ADC    X4
         ADD    (2),X4
R26.7:
         TST    (6)+                     ;IGNORE EXPONENT
         JMP    MOD                      ;TO MODULUS,NORMALISE AND EXIT
;
;
         .PAGE
         .SBTTL FLOATING MULTIPLY, BASIC
;
;  FLOATING POINT MULTIPLY
```

```
;
R28:                                                    (0-8-18)

R28.0:
          JSR    X3,FPENT                  ; USE COMMON ENTRY ROUTINE
         TST      X1
         BEQ      UFLO
         TST      X4
         BEQ      UFLO
         MOV      X5,-(6)
         ADD      X4,X1
         SUB      #202,X1
         MOV      X1,-(6)
         CMP      -(2),-(2)
         MOV      (2)+,X1
         MOV      (2)+,X5
         ASL      2(2)
         ROL      (2)
          CLR    X4                        ; CLEAR X4 AND X3 TO HOLD RESULT
          CLR    X3
R28.1:    ASR    X4                        ; SHIFT RESULT ONE PLACE RIGHT
          ROR    X3
          ASR    X1                        ; PICK UP BOTTOM BIT OF
          ROR    X5                        ; MULTIPLIER AND LEAVE SHIFTED
                                           ; ONE PLACE RIGHT
          BCC    R28.1                     ; NOTHING TO DO THIS TIME
          ADD    2(2),X3                   ; ADD MULTIPLICAND INTO RESULT
          ADC    X4
          ADD    (2),X4
          TST    X5                        ; MORE TO DO IN MULTIPLIER?
          BNE    R28.1
          TST    X1
          BNE    R28.1
          BR     PREXIT                    ; USE COMMON EXIT ROUTINE
;
;
        .PAGE
        .SBTTL FLOATING DIVIDE, BASIC
;
;
; FLOATING POINT DIVIDE
;
R29:
R29.0:
          JSR    X3,FPENT                  ; USE COMMON ENTRY ROUTINE
          TST    X1                        ; DIVIDE BY ZERO?
          BEQ    OVFLO                     ; OVERFLOW
          TST    X4                        ; CLEAR X4 AND X3 TO HOLD RESULT
          BEQ    UFLO
          SUB    X1,X4                     ; CALC RESULT EXPONENT
          ADD     #200,X4                   ;ADJUST SCALING
          MOV    X5,-(6)                   ; DUMP
         MOV      X4,-(6)
         CLR      X3
         CLR      X4
          MOV    (2)+,X1                   ;PICK UP DIVIDEND
          MOV    (2),X5
          CMP     -(2),-(2)                          ;POINT X2 AT DIVISOR
          CMP    -(2),X1                    ; IF DIVISOR > DIVIDEND THEN
          BLT    R29.1                     ; SHIFT DIVIDEND
          BGT    R29.8
          CMP    2(2),X5
```

0-8-19

```
          BLOS   R29.1                    ;TEST BOTTOM HALVES
R29.8:    ASL    X5                       ; SHIFT DIVIDEND
          ROL    X1
          DEC     (6)
R29.1:    CLR    -(6)                     ; SET 2 WORDS ON STACK TO BE
          MOV     #400,-(6)                ;THE BIT TO ADD TO THE RESULT
R29.2:    CMP    X1,(2)                   ; IF DIVIDEND > DIVISOR THEN
          BGT    R29.9
          BLT    R29.3                    ; SUBTRACT DIVISOR AND PUT BIT
          CMP    X5,2(2)                  ; INTO RESULT
          BLO    R29.3                    ;TEST BOTTOM HALVES IF TOPS =
R29.9:    BIS    (6),X4                   ; SET BIT INTO RESULT
          BIS    2(6),X3
          SUB    2(2),X5                  ; DOUBLE LENGTH SUBTRACT
          SBC    X1
          SUB    (2),X1                   ; IF DIVIDEND IS ZERO AFTER THE
R29.3:    ASL    X5                       ; SHIFT DIVIDEND UP ONE
          ROL    X1
          ASR    (6)                      ; SHIFT 'BIT' DOWN ONE
          ROR    2(6)
          BNE    R29.2                    ; IF 'BIT' IS SHIFTED RIGHT OUT
          TST    (6)                      ; OF THE DOUBLE WORD THEN IS
          BNE    R29.2                    ; NO MORE TO DO
R29.4:    CMP    (6)+,(6)+                ; PICK UP EXPONENT
PREXIT:
          MOV    (6)+,X1
          MOV     (6)+,X5
          JMP     ALIGN
;
;
;
;
.ENDC                     ;(ZFPIL)
          .PAGE
          .SBTTL FLOATING ABS & NEG, COMMON
.IFTF                     ;(Z45FP)
;
; FLOATING POINT ABS
;
;
R30:
R30.0:    BIC     #100000,2(6)             ;CLEAR SIGN BIT
          RTS    X2
;
;
;
;
;
; FLOATING POINT NEGATE
;
R31:
R31.0:    TST     2(6)                     ;DO NOT NEGATE IF
          BEQ     R31.1                    ;CONVENTIONAL ZERO
          ADD     #100000,2(6)             ;FLIP SIGN BIT
R31.1:
          RTS    X2
;
          .PAGE
;
.IFT                      ;(Z45FP)
;
          .SBTTL FLOATING COMPARE, BASIC
```

```
;
; FLOATING POINT COMPARE
;
R32:

R32.0:  TST    (6)+                    ; IGNORE OLD X2
        MOV    (6),X1                  ; PICK UP M.S. HALF OF FIRST
                                       ; REAL TO USE AS RESULT FLAG
        BIS     #200,X1                ;PUT IN 'NORMALISE' BIT TO
                                       ; DEAL WITH R1=0.0 CASE
        ASL    4(6)                    ; LOOK AT SIGN BIT OF SECOND REAL
        BEQ    R32.5                   ; LOOK FOR -0.0
        BCS    R32.2                   ; BRANCH IF NEGATIVE
        ; SECOND REAL IS >=0
R32.5:  ASL    (6)                     ; LOOK FOR SIGN BIT OF FIRST REAL
        BEQ    R32.1                   ; LOOK FOR -0.0
        BCS    R32.3                   ; BRANCH IF NEGATIVE AS NOW
                                       ; KNOW RESULT OF COMPARE
R32.1:  ; BOTH REALS HAVE SAME SIGN
        CMP    4(6),(6)                ; COMPARE M.S. HALVES
                                       ; TREAT AS UNSIGNED INTEGERS
        BHI    R32.4                   ; R2 FRACTION > R1 FRACTION
        BLO    R32.3                   ; R2 FRACTION < R1 FRACTION
        CMP    6(6),2(6)               ; DITTO FOR L.S. HALVES
        BHI    R32.4
        BLO    R32.3
        CLR    X1                      ; REALS ARE EQUAL SO CLEAR
        BR     R32.4                   ; RESULT FLAG
R32.2:  ; SECOND REAL IS <0
        ASL    (6)                     ; LOOK AT SIGN BIT OF FIRST REAL
        BCC    R32.3                   ; R2<0, R1>=0 SO KNOW RESULT
        BR     R32.1                   ; BOTH NEGATIVE
R32.3:  COM    X1                      ; CHANGE SIGN OF RESULT
R32.4:  ADD     #8.,X6                 ;SKIP OVER OPERANDS
        TST    X1                      ; SET CONDITION CODES
        JMP    (2)                     ; RETURN
                                       ; SIZE - 30 WORDS
                                       ; AVERAGE TIME ON 11/20 -
                                       ; IF SIGNS DIFFER - 30
                                       ; IF SIGNS SAME - 45
                                       ; STACK USAGE - 5+0 WORDS
        .PAGE
        .SBTTL CONVERSIONS TO REAL, BASIC
;
; TYPE CONVERSION ROUTINES
;
;
; INT TO REAL
; ASSUMES ONE WORD IS ON THE STACK
;
R33:
R33.0:  MOV     #210,X1                ;SET INITIAL EXPONENT
R33.1:  MOV    2(6),(6)                ;) CONVERT TO
        CLR    2(6)                    ;) BIGINT
        TST    -(6)                    ; ADJUST STACK POINTER
        BR     R35.1                   ; FLOAT AS A BIGINT
;
;
;
; FRAC TO REAL
; SETS INITIAL EXPONENT AND USES INT TO REAL ROUTINE
;
```

```
R34:
R34.0:    MOV     #171,X1                                    (0-8-21)
          BR      R33.1
;
;
;
;
; BIGINT TO REAL
; ASSUMES DOUBLE WORD IS ON STACK
;
R35:
R35.0:    MOV     #230,X1                    ;SET INITIAL EXPONENT
R35.1:
          TST     -(6)
          MOV     X2,-(6)
          MOV     X3,-(6)                    ; DUMP X3 AND X4
          MOV     X4,-(6)
          MOV     10.(6),X4                   ; PICK UP DOUBLE WORD
          MOV     12.(6),X3
MOD:
          TST     X4                         ; NEGATE IF NEGATIVE
          BGE     R35.2
          NEG     X3                         ; DOUBLE LENGTH NEGATE
          ADC     X4
          NEG     X4
          BVC     R35.3                       ; ADJUST FOR MOST NEG NUMBER
          ADC     X1
          ROR     X4
R35.3:
          SEC
R35.2:    ROR     -(6)
          JMP     ALIGN
;
;
;
;
;  MIXED TO REAL
; SETS INITIAL EXPONENT AND USES BIGINT TO REAL ROUTINE
;
R36:
R36.0:    MOV     #211,X1
          BR      R35.1
;
;
;
;
; FINEFRAC TO REAL
; SETS INITIAL EXPONENT AND USES BIGINT TO REAL ROUTINE
;
R37:
R37.0:    MOV     #172,X1
          BR      R35.1
;
;
          .PAGE
;
.IFTF                       ;(Z45FP)
          .SBTTL CONVERSIONS FROM REAL, COMMON
;
; R38 AND R39 COMMON TO 11/45 SET AND BASIC SET
;
;
```

```
; REAL TO FRAC
;
R38:
R38.0:   MOV     #201,X1          ;SET INITIAL EXPONENT
R38.1:   MOV     X3,(6)                  ;DUMP X3 AND X4
         MOV     2(6),X3
         ROL     X3                     ; SHIFT UP EXPONENT AND STORE
         CLRB    X3                     ; BRING EXPONENT TO BOTTOM
         SWAB    X3                     ; OF WORD
         SUB     X1,X3                  ; X1 IS NOW SHIFT COUNT
         MOV     #100000,X1              ;X1 BECOMES LARGEST -VE NO.
         BCC     R38.8
         CMP     #-16.,X3
         BGT     R38.7
         SWAB    4(6)                    ;PICK UP REAL AS FRACTION
         MOVB    2(6),5(6)               ;BOTTOM BYTE.REPLACE NORMALISE
         BIS     4(6),X1                 ;BIT.(SWAB CLEARED CARRY)
         TST     2(6)                    ;TEST SIGN
         BGE     R38.4                   ;SKIP IF +VE
         NEG     X1                      ;NEGATE IF NEGATIVE
R38.4:   ROR     X1                      ;FIRST SHIFT DOWN.X1 NOW +VE
         BR      R38.3                   ;IN SUBSEQUENT SHIFTS.THE CARRY
R38.2:   ASR     X1                      ;IS NOT USED
R38.3:   INC     X3                      ;COUNT SHIFT
         BLT     R38.2                   ;MORE SHIFTS TO DO
         ADC     X1                      ;ROUND CAN CAUSE OFLO ON MOST
R38.5:   BVC     R38.6                   ;POSITIVE NUMBER
         COM     X1                      ;MOST -VE BECOMES MOST +VE
R38.6:   MOV     (6)+,X3                 ;RESTORE X3
         TST     (6)+                    ;SKIP OLD MS PART
         MOV     X1,(6)                  ;STORE RESULT
         JMP     (2)                     ;RETURN
R38.7:   CLR     X1                      ;SET ZERO FOR UNDERFLOW
         BR      R38.6
R38.8:   CMP     X1,2(6)                 ;SET OFLO IF REAL IS +VE
         BR      R38.5                   ;SINCE X1 IS MOST NEGATIVE
;
;
;
;
; REAL TO INT
; SETS INITIAL EXPONENT AND USES REAL TO FRAC ROUTINE
;
R39:
R39.0:   MOV     #220,X1
         BR      R38.1
;
         .PAGE
.IFF                        ;(Z45FP)
         .SBTTL FLOATING COMMON ENTRY,11/45
;
; 11/45 FP ROUTINES
;
; COMMON 11/45 FLOATING POINT ENTRY ROUTINE
;
FPENT:
         LDFPS   #0         ;LOAD FPP PROGRAM STATUS
         MOV     (6),2(6)
         TST     (6)+
         RTS     X3
;
; FP ADD
```

```
;
R26:
R26.0:
        JSR     X3,FPENT        ;INITIALISATION
        LDF     (X6)+,AC0       ;2ND OPERAND
        ADDF    (X6),AC0        ;ADD FIRST OPERAND
;
        .SBTTL FLOATING OFLO & EXIT, 11/45
; COMMON OVERFLOW CHECK
;
FPOFLO:
        MOV     #FPMAX,X1
        CFCC            ;SET CPU'S CC
        BVC     R26.9   ;NO OVERFLOW
FPOFL3:
        BPL     R26.5   ;BR IF +VE
        TST     (1)+
R26.5:
        LDF     (1),AC0 ;SET MAXIMUM + OR - RESULT
R26.9:
        STF     AC0,(X6)        ;STACK RESULT
        JMP     (2)
;
FPMAX: .WORD 077777,177777,177777       ;MAXIMUM +VE AT FPMAX,FPMAX+2
                                        ;MAXIMUM -VE AT FPMAX+2 & 4
        .PAGE
        .SBTTL FLOATING ADD SUB MUL DIV COMP,11/45
;
; FP SUBTRACT
;
R27:
R27.0:
        JSR     X3,FPENT        ;INITIALISATION
        LDF     (X6)+,AC1       ;2ND OPERAND
        LDF     (X6),AC0        ;1ST OPERAND
        SUBF    AC1,AC0         ;PERFORM SUBTRACTION
        JMP     FPOFLO          ;EXIT AS ADD
;
; FP MULTIPLY
;
R28:
R28.0:
        JSR     X3,FPENT
        LDF     (X6)+,AC0       ;SECOND OPERAND
        MULF    (X6),AC0        ;TIMES FIRST
        JMP     FPOFLO          ;EXIT AS ADD
;
; FP DIVIDE
;
R29:
R29.0:
        JSR     X3,FPENT        ;INITIALISATION
        LDF     (X6)+,AC1       ;DIVISOR
        CFCC
        BNE     R29.3   ;BRANCH IF NOT DIVIDE BY ZERO
        LDF     (X6),AC0        ;DIVIDEND
        CFCC
        JMP     FPOFL3
R29.3:
        LDF     (X6),AC0        ;LOAD DIVIDEND
        DIVF    AC1,AC0         ;DO DIVISION
        JMP     FPOFLO
```

```
;
; FP COMPARE
;
R32:
R32.0:
        JSR     X3,FPENT        ;INITIALISATION
        LDF     (6)+,AC0        ;SECOND OPERAND
        CMPF    (6)+,AC0                ;COMPARE WITH FIRST
        CFCC
        JMP     (2)
        .PAGE
        .SBTTL CONVERSIONS TO REAL, 11/45
;
; INT TO REAL
;
R33:
R33.0:
        JSR     X3,FPENT        ;INITIALISATION
        LDCIF   (6),AC0
        TST     -(6)            ;SPACE FOR RESULT
        STF     AC0,(6)         ;RESULT TO STACK
        JMP     (2)
;
; FRAC TO REAL
;
R34:
R34.0:
        JSR X3,FPENT    ;INITIALISATION
        LDCIF   (X6),AC0
        STEXP   AC0,-(6)        ;STORE EXPONENT
        SUB     #15.,0(6)       ;REDUCE FOR FRAC ALIGNMENT
        LDEXP   (6),AC0         ;ADJUSTED EXPONENT BACK INTO NO.
        STF     AC0,(6)         ;STORE FINAL ANSWER
        JMP     (2)
;
; BIG INT TO REAL
;
R35:
R35.0:
        JSR     X3,FPENT        ;INITIALISATION
        SETL            ;LONG MODE
        LDCLF   (X6),AC0
        STF     AC0,(6)
        JMP     (2)
;
; FINE INT OR BIG FRAC TO REAL
;
R36:
R36.0:
        JSR     X3,FPENT        ;INITIALISATION
        SETL            ;LONG MODE
        LDCLF   (6),AC0
        SETI
        STEXP   AC0,-(6)
        SUB     #15,0(6)        ;ADJUST EXPONENT FOR FRAC
        LDEXP   (X6)+,AC0
        STF     AC0,(6)
        JMP     (X2)
;
; FINE FRAC TO REAL
;
R37:
```

0-8-25

```
R37.0:
        JSR     X3,FPENT        ;INITIALISATION
        SETL
        LDCLF   (X6),AC0
        SETI
        STEXP   AC0,-(X6)
        SUB     #30.,0(6)
        LDEXP   (X6)+,AC0
        STF     AC0,(6)
        JMP     (2)
.ENDC                   ;(Z45FP)
.ENDC                   ;(ZFLPT)
LENGTH:         ;SO WE CAN SEE LENGTH ON SYMBOL TABLE
SMT:    JMP     SMT...
        .END    SMT
```

SMTUIX.EDT:

```
=MAIN .
F 'LET GO'
I
LET NFREETASKS  = 14;                    % NUSERTASKS-USEDTASKS          %
^Z
S/DEFIN/DEFIN,INTTY/'DEFIN;'
S/DEFOUT/DEFOUT,OUTTTY/'DEFOUT;'
S/ERPRIN;/ERPRIN,MULTI,COM;/'EXT PROC () FBPROC'
S/;/,MULTISTK,COMSTK;/'EXT STACK FBSTK'
S/FBSTK(NUSERT/MULTISTK,COMSTK,FBSTK(NFREET/'FBSTK(NUSER'
S/0(NUSERT/20,10,0(NFREET/'0(NUSERTASKS)'
S/NOGO(NUSERT/GO,GO,NOGO(NFREET/'NOGO(NUSERTASKS)'
D 'USAGE:=0;'
D 'TITLE SMT INTERACTIVE DEVICE DRIVER':%E
```

SMTBIX.EDT:

```
=MAIN .
S/28/26/'+++ 26 FOR LSI'
S:MOV *UPS/TRAPDATA,177776:MTPS *UPS/TRAPDATA:'RESTORE USERS PS'
S/158/152/'+++ 152 FOR LSI'
S/MOV @#177776,/MFPS /'@#177776'
S:MOV #*MASK,@#177776:MTPS *HIPRI/TASKDATA:'+++ MTPS'
S:MOV #*UNMSK,@#177776:MTPS *LOPRI/TASKDATA:'+++ MTPS'
S/2/0/'+++ 0 FOR LSI'
S:MOV   #*MASK,@#177776:MTPS *HIPRI/TASKDATA:'+++ MTPS'
S/2/0/'+++ 0 FOR LSI'
S:MOV   #*UNMSK,@#177776:MTPS *LOPRI/TASKDATA:'+++ MTPS'
S/62/60/'+++ 60 FOR LSI'
S:MOV #*MASK,@#177776:MTPS *HIPRI/TASKDATA:'+++ MTPS'
S/48/44/'+++ 44 FOR LSI'
S:MOV   @#177776,:MFPS   :'++ MFPS'
S:MOV   #*MASK,@#17776:'MTPS *HIPRI/TASKDATA:'+++ MTPS'
```