


2

AN INTELLIGENT MAGNETIC TAPE CONTROLLER


ALEXANDER DONALD MCGUFFOG

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

September 1986



The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

ACKNOWLEDGEMENTS

Tony Eva and Graham Jack, for assistance in the use of the PDP-11.

Derek Sherlock, who assisted in several aspects of the production of this thesis.

The CSIR, for their financial assistance.

ABSTRACT

This thesis describes a system to allow a mass storage device to be installed in a position remote from the computer system which controls it. This system is intended to allow undergraduate students in the Electrical Engineering department at UCT to make use of two nine channel tape drives installed in the undergraduate laboratory for project work. The drives are interfaced to the department's PDP-11/23 computer, and may be accessed by standard operating system directives, as the controller simulates a conventional computer peripheral.

The system consists of an SA-Bus based tape transport controller which interfaces to the host computer system via a serial line. The following hardware was designed and built specifically for this system :

1. A CPU card based on the Intel 80188 microprocessor, incorporating high speed DMA (direct memory access) channels and two interrupt driven serial lines.
2. A timing and control module for the tape transports. This consists of two SA-Bus cards.

Two sets of software were written for the system. These are the following :

1. Software to operate the tape controller. This consists of six modules written in Pascal-86 and 8086 assembler.

2. Software to allow the PDP-11/23 to control the the tape drives. This is in the form of an RSX-11 device driver written in PDP-11 assembler.

To allow the system to be easily upgraded in the future (in particular to allow the system to be incorporated into UCT's proposed local area network), the software was written in a highly modular form.

In addition to being controlled by a host system in remote mode the tape controller also has the ability to perform a variety of operations in local mode. These include the ability to copy and erase tapes, as well as a comprehensive set of diagnostic functions. When in local operations mode the controller is menu driven, making its use by persons who are not familiar with it quick and easy.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

ABSTRACT

TABLE OF CONTENTS

. CHAPTER 1: INTRODUCTION

CHAPTER 2: THE TAPE TRANSPORTS

CHAPTER 3: THE 80188 CPU CARD

CHAPTER 4: THE TAPE INTERFACE MODULE

CHAPTER 5: THE RSX-11M OPERATING SYSTEM

CHAPTER 6: THE DEVICE DRIVER

CHAPTER 7: THE TAPE CONTROLLER SOFTWARE

CHAPTER 8: CONCLUSION

REFERENCES

APPENDIX A: USER GUIDE

APPENDIX B: CIRCUIT DIAGRAMS : CPU CARD

APPENDIX C: CIRCUIT DIAGRAMS : TAPE INTERFACE MODULE

APPENDIX D: DATA LINK FORMAT INFORMATION

APPENDIX E: SOFTWARE LISTINGS : PDP-11 DEVICE DRIVER

APPENDIX F: SOFTWARE LISTINGS : TAPE CONTROLLER SOFTWARE

APPENDIX G: SOFTWARE BUILD INFORMATION

APPENDIX H: PASCAL-86 V3 FAULTS

CHAPTER 1

INTRODUCTION

1.1. Requirements

This project arose out of the need of the University of Cape Town's department of Electrical and Electronic Engineering for an interface which will allow a mass storage device for the department's PDP-11/23 minicomputer system to be installed in a position remote from the central processing unit's position.

The PDP-11 is used for various purposes, notably for undergraduate teaching and postgraduate research. It is from its role as undergraduate teaching tool that this requirement arose. Undergraduate teaching is done mainly on the department's so called SABUS kits. These are SABUS [8],[9] based microcomputer systems designed by UCT specifically for the purpose of undergraduate teaching. The kits are installed in a separate undergraduate laboratory and are connected to the PDP-11 via serial lines.

The kits function as work stations. They are used for editing source files for processing on the PDP-11, as well as for assembly and execution of 8085 code, which forms a key part of the current undergraduate programme. All long term file storage is however done by the PDP-11 system, normally on fixed disks. It is with this file storage that severe problems can arise. At certain times of the year several courses, each with up to 80 students, make use of the kits, and hence the PDP-11 system for file storage. This can result in severe storage space problems, especially as RSX-11/M, the operating system used on the PDP-11, maintains copies of each version of a disk file unless these are explicitly deleted. Should space problems occur these cannot be rectified by the students themselves, but only by the actions of a user with a privileged account (normally a member of staff). At times of peak demand the system must be available 24 hours a day. A form of mass storage to relieve this congestion is a high priority for the department.

1.1.1. In practice, two possible ways exist to reduce the space problems :

1.1.1.1. Increase the available storage space. This approach was rejected as firstly storage for PDP-11 systems is extremely expensive compared to other forms of mass storage, and secondly that a very large amount of such storage would be required to ensure that the problems did not arise. The exact amount of storage required is highly dependant on the type of work being done, but for most multi-user systems is estimated at several megabytes of storage per user. This would imply a requirement of well in excess of 100 Mbytes of disk storage, which would be difficult to justify as it is only required at certain times of the academic year.

1.1.1.2. Make some form of removable medium mass storage available to the students. Using this system, each student (or group of students) can be issued with their own storage medium (magnetic tape or floppy disk for example). The student's files will then not need to be stored on the PDP-11's disks.

1.1.2. For the university's purposes the second solution is far better, for reasons of cost and simplicity. It should be noted that the specific problem described above is only one example of a more general problem facing the data processing industry. Traditionally computer systems have consisted of a CPU (Central Processing Unit) with its peripherals in close proximity to it (normally in the same room). The only peripherals installed in remote locations were terminals and occasionally line printers and card readers. With the recent dramatic advances in integrated circuit technology there has been a trend towards distributed systems. Distributed systems differ from the traditional computer system architecture by having the intelligence in the system spread throughout the entire computer system rather than being concentrated in a single CPU. In practice these systems normally consist of a number of work stations and a central file server (concentration of mass storage devices). It has however been found that it is desirable to have some form of mass storage accessible to each user. This has the following advantages :

1.1.2.1. Each user can keep his own copies of important files. The system manager no longer has to make frequent back-ups of all files. This results in both lower overheads and a greater feeling of security on the part of the user.

1.1.2.2. The user can easily remove files not in use from the system. If the user has no easy way of rolling files which are not immediately required out of the system the

disk drives will quickly become cluttered up with unused files.

1.1.2.3. It is simpler for the user to import files from another similar system. This form of information interchange is often important, especially in organizations which carry out activities in which data from external sources plays a large role (for example research establishments).

1.1.3. As a result of these considerations it was decided to construct an interface which would allow a mass storage device for the PDP-11 to be installed in the department's digital undergraduate laboratory. The actual mass storage device chosen was a pair of 800 BPI (Bits Per Inch) NRZ (Non Return to Zero) nine channel magnetic tape drives. This was chosen for the following reasons :

1.1.3.1. Two such tape transports were available from scrapped equipment donated to the department.

1.1.3.2. The de facto industry standard for magnetic tapes used for program transfer is 800 BPI. Although the department has a tape transport attached to the PDP-11, this is a 1600 BPI phase encoded drive which cannot read 800 BPI tapes. As a result in the past tapes have had to be converted from 800 BPI to 1600 BPI by UCT's computer center, a process which is both time consuming and prone to problems as a result of the wide variety of tape formats used on PDP-11 systems. The ability to read 800 BPI tapes was thus a further attraction of the project.

1.2. Description of the Tape Transports

Both tape transports conform to the de facto industry standard design. They have the following features in common :

1.2.1. Industry standard control bus. This consists of three connectors, one for write data, one for read data and one for control lines. In practice this interface is not completely standardized, and variations exist between manufacturers, especially as regards methods of selecting different units on a common bus.

1.2.2. The transports provide encoding and decoding of the NRZI recording code, as well as clock recovery.

1.2.3. An interface is required, external to the unit, which must provide the following functions :

1.2.3.1 All tape formatting. This consists of parity information (one bit per character) and a 9 bit CRC (Cyclic Redundancy Check) for error control.

1.2.3.2 All timing. All data is written in the form of blocks. Recognition of these blocks by the controller is by timing (for example a CRC is preceded by 3 character spaces). It is the responsibility of the interface unit to provide all timing both for write and read operations.

1.3. The data link

The primary objective of this project is to design an interface for industry standard magnetic tape drives which is suitable for use with the PDP-11 and will allow the tape transports to be located at a distance from the main computer system. For this reason the choice of a data link is critical to the project. Conventional mass storage devices are interfaced to the host system via parallel data busses which are suitable only for transmission over a maximum distance of only a few meters. For this project a system is required which is capable of transmitting over a distance of several hundred meters. Two techniques are in common use [15], and meet this requirement. These are the following :

1.3.1.1. A LAN (Local Area Network) system. This is a system which allows several devices to be connected on a single high speed line (normally a twisted pair or co-axial cable). This kind of system can reach data transfer rates of 10 Mbits/s.

1.3.1.2. Conventional serial lines. These are normally RS-232, and operate at speeds of up to 9600 baud (Baud refers to the number of changes per second on a line. For a conventional line this is almost the same as bits per second.), or in exceptional cases 19200 baud.

In this case it was decided to use serial RS-232 lines. This was done for the following reasons :

1.3.2.1. The PDP-11 system has serial lines available which are not in use.

1.3.2.2. Although UCT's proposed LAN system will interface to the PDP-11 system, and operates at a far greater speed than a conventional serial line, the hardware for this system did not exist at the time. In addition the software for the LAN has not been finalized to date. It was however decided to design the tape interface in such a way as to make an upgrade to operation on the LAN as easy as possible.

1.4. PDP-11 software

The physical interface to the PDP-11 is via a serial line. The software used on the PDP-11 in order to operate the tape controller will now be discussed. The operating system used on the PDP-11 system is RSX-11M. This has the following features :

1.4.1. It is multi-user. This means that several user can use the system simultaneously.

1.4.2. It is multi-tasking. This means that several programs can be run simultaneously, even by the same user.

1.4.3. It is hierarchical. From the first two points mentioned above it is clear that strict control is necessary over system resources, such as peripherals. A situation could easily arise in which two or more programs wished to use a peripheral at the same time. The hierarchical nature of the RSX-11M operating system allows it to keep strict control over the various peripherals in the system. The operating system may be viewed as a series of layers, with a user's application program as the first layer, the various parts of the operating system as the succeeding layers culminating in the layer which actually controls the

peripheral. The components of this layer are known as device drivers, and one exists for each type of device in the system. The RSX-11 operating system makes special provision for the user to include his own drivers at this level, and so customize the system to his own requirements. As the driver executes as part of the operating system strict rules are associated with it in order to ensure the integrity of the operating system. It is via this mechanism that the tape transports are controlled.

1.5. Other Requirements

In addition to the requirements discussed above various other features would be desirable :

1.5.1. The interface unit should be as easy as possible to incorporate into the proposed LAN system. As LAN cards are being designed for UCT's SABUS based systems the tape controller interface should be SABUS based. This has the additional advantage that card cages with built-in power supplies are readily available, as are memory cards.

1.5.2. The interface unit should be as intelligent as possible. The host computer should have to do the absolute minimum of control. For example if an error is detected on a read operation the controller should automatically retry the operation.

1.5.3. The interface unit should be able to perform as many operations as possible in local mode. If the operator can perform certain operations, for example to copy a tape, using only the controller then a great deal of time will be

saved. It was therefore decided to incorporate a second serial port into the controller to interface to a terminal. This allows the interface to be directly controlled by the user.

1.5.4. The interface should have as many diagnostics functions as possible available. This not only simplifies testing of the system, but also will make the system a great deal easier to modify, if required, at a later date.

1.5.5. The interface should emulate a known tape transport/controller combination. This has several advantages :

1.5.5.1.1. The interface will be compatible with as much DEC operating system software as possible. This is an extremely important consideration. From the point of the user of the PDP-11 the tape controller should appear to simply be a standard DEC peripheral.

1.5.5.1.2. Source code of the device drivers for these devices is available. This simplifies the task of ensuring that the tape controller subsystem is compatible with the operating system.

1.5.5.1.3. The behavior of the controller can be compared against known, documented equipment.

Although several of these are manufactured by DEC (Digital Equipment Corp.(manufacturer of the PDP-11)) the TM11 controller was chosen. This was for the following reasons :

1.5.5.2.1. The TM11 is the standard controller for the type of tape transports which are available.

1.5.5.2.2. The TM11 controller is the basis of the entire DEC range of tape controllers. Although the various controllers all have different capabilities they all preserve TM11 compatibility. Most DEC tape software uses only TM11 commands, and so ensures compatibility with all tape available interfaces.

1.5.5.2.3. The tape interface in use to control the department's 1600 BPI phase encoded drive is a hardware emulation of the TM11 controller. This means both that direct comparisons can be done and that special purpose software written at UCT specifically for this tape drive will run without modification.

1.6. The Tape Controller Hardware

In order to meet the requirements set out above an SABUS based tape controller interface was constructed. This consisted of the following subsections :

1.6.1. Card cage and power supply. This is a conventional 19-inch rack mounting box and card cage with built-in power supply identical to those used for various other UCT systems.

1.6.2. CPU card. This card which was especially designed and constructed for this project and has the following features :

1.6.2.1. 80188 CPU. This is an advanced INTEL microprocessor with various peripherals integrated on chip (for example timers and a DMA (Direct Memory Access) controller). The instruction set of this microprocessor chip is a superset of the popular 8088/8086 processor. The advanced design of this processor allowed a considerable reduction in size of the hardware compared to other microprocessors while allowing the use of 8088/8086 development equipment, notably Pascal-86, a high level language used for much of the controller software.

1.6.2.2. EPROM storage on board. Provision was made for up to 128 Kbytes of long term program storage on the CPU card.

1.6.2.3. Serial interface lines. Two serial lines were included on the CPU card for the data link and terminal. Provision was made for both to be interrupt driven in order to allow the highest possible data transfer rate.

1.6.3. Interface module. This module, also designed and constructed especially for this project provides the physical interface to the tape transports, as well as all timing functions. It consists of two SABUS cards connected via a ribbon cable. Data transfers are via DMA.

1.6.4. A memory module. This is a standard 128K dynamic RAM card as used in UCT's SABUS kits. It has 64 Kbytes of memory installed.

1.7. System Software

Two sets of software are required :

1.7.1. Controller software. This is the software which is embedded in the controller. This consists of five modules, of which two are written in Pascal-86, and the rest in assembler. These modules provide all controller functions.

1.7.2. PDP-11 software. The software for the PDP-11 consists simply of a device driver (known as MA drive) which emulates a TM11 tape controller as discussed above. This is written in MACRO-11, the PDP-11 assembler language.

CHAPTER 2

THE TAPE TRANSPORTS

2.1. Introduction

In chapter 1 the objectives of this project were discussed, and it was stated that two nine channel tape drives were selected to provide a remote mass storage system. These tape drives, and the formats used to record data on them, will now be discussed. Both tape transports are of industry standard design, although in practice there are minor differences between them. The tape transports are designed to read IBM compatible tapes, which are the industry standard for data transfer. The requirements for IBM compatibility are described below :

2.2. IBM Magnetic tape standard

2.2.1. IBM compatibility means that the tape is readable by an IBM 2400 series tape transport and the converse. It should however be noted that this this compatibility extends

only to the level of physical compatibility, that is parameters such as number of tracks, track width, form of check characters, length of inter block gaps, recognition of file gaps etc. These parameters are designed only to ensure that the hardware in a particular system is capable of reading and writing standard tapes. Any other formatting, for example at file and volume level, is not specified.

2.2.2. The IBM standard covers both 7 channel (6 bits of data and one of parity) and 9 channel (8 bits of data and one of parity). Although the tape transports used for this project support only 9 channel operation both formats will be described as the tape controller is capable of supporting both 7 and 9 channel operation.

2.2.3. The method used for recording is that of NRZl (non-return-to-zero change at logic 1), also often simply referred to as NRZ (non-return-to-zero). Recording using this method of encoding is simple : Whenever a logical one occurs in the bit stream for a particular track, the direction of the head current, and hence the tape magnetization, is changed. A special character called the LRC (Longitudinal Redundancy Character) ensures that the direction of magnetization is always the same in the inter-record gaps. The use of this character will be discussed further in section 2.2.5.3. NRZ has two advantages over other recording techniques such as phase encoding or group coded recording [24]:

2.2.3.1.1. Head current flows at all times. This makes it possible to record over old data. In practice most tape transports include an erase head to ensure tape erasure in the inter-track gaps. This significantly

reduces problems caused by badly aligned heads and hence improves reliability.

2.2.3.1.2. The electronics for reading and writing NRZl tapes are simpler than for other techniques [24].

2.2.3.2. A significant disadvantage is, however, that unlike the two other techniques mentioned above, NRZ is not self-clocking and is hence useful only where multiple track recording is employed or limits are placed on the data content. A further restriction is that in order to recover a character clock at least one channel must have a one recorded in it. This may be ensured by employing odd parity. If odd parity is not employed then the all zeros character must be declared invalid. Odd parity is always employed on 9 channel tapes. Note that phase encoding and group coded recording allow the use of phase lock loop type data recovery circuits, allowing both higher tape density and the recording of any data pattern.

2.2.3.3. Writing NRZl tapes. In order to record data on a magnetic tape it is necessary to magnetize the tape discretely to indicate binary ones and zeros. In the NRZl method current flows in the write head continually while the transport is in the write mode. Binary ones are indicated by a reversal in the direction of current flow, and hence of the saturated magnetism on the tape. It is the responsibility of the user to provide the data and the write clock. Actual encoding is done by the tape transport electronics.

2.2.3.4. Reading NRZl tapes. In order to recover data written as described above the tape is passed over a read head at constant velocity. When the resulting playback

signal is rectified a pulse is present for each one recorded in the original wave form. If a zero was recorded no pulse will be generated, and the clock to store the data must be obtained from another channel.

2.2.4. The IBM standard also specifies various tape parameters, such as skew between channels, gap scatter and position of the tracks on the tape. As these specifications are determined by factors such as head design which are the concern only of the manufacturer of the tape transport they will not be discussed here. Readers wishing for further information should consult references [1] and [2].

2.2.5. Tape format. This refers to the methods used to delimit blocks of data, the beginning and end of the tape and the check codes used to ensure data integrity. Two structures may be written to the tape. These are data records (or blocks) which consist of data characters and check characters, and file marks, which delimit sections of data. The format of these structures will be described later. The following factors define the tape format:

2.2.5.1. Tape markers. To avoid physical damage to the recorded surface a portion of tape at the beginning and end of the tape is reserved for threading and loading. These sections are delimited by reflective markers on the non-oxide side of the tape. The markers are referred to as the BOT (Beginning Of Tape) and EOT (End Of Tape) markers respectively. These gaps are at least 10 feet, and the tape is erased for at least 1.7 inches before the trailing edge of the BOT tab. Note that it is permissible to write over and beyond the EOT marker.

2.2.5.2. Gaps. Reading and writing can take place reliably only when the tape is moving at a constant velocity. To allow the tape to start and stop, gaps must be left between data records. The following gap lengths are used :

2.2.5.2.1. Beginning of tape gap. An erased section of tape is required surrounding the BOT marker. This extends for at least 1.7 inches ahead of the trailing edge of the marker and 0.5 inches past it.

2.2.5.2.2. Inter record gaps. These gaps are between data records, and have a minimum length of 0.75 inch for seven-track and 0.6 inch for nine-track.

2.2.5.2.3. End of file gaps. This gap is a minimum of 3.9 inches for seven-track and 3.75 inches for nine-track.

2.2.5.3. Data record format. Data records consist of a number of characters and a number of check characters. A minimum of 14 characters and a maximum of 8192 characters may be written in a single block. For seven-track tape a single check character is written. This is the LRC (Longitudinal Redundancy Character), and is spaced four character spaces from the last data character. For nine-track tape a CRCC (Cyclic Redundancy Check Character) is written in addition to the LRC, spaced four character spaces after the last data character. The LRC is spaced four character spaces after the CRCC. The check characters are obtained as follows :

2.2.5.3.1. The LRC. The polarity of each bit in the LRC is such that the number of magnetic transitions in each track is even. This ensures that the direction of the magnetic saturation is the same in all tape gaps.

2.2.5.3.2. The CRCC. A complete discussion of cyclic codes is too lengthy to present in this document. An introduction to the subject may be found in references [3],[4] and [5], and only information which is pertinent to the tape format is given here. The following information is specific to the code used in 9-track recording.

1. The parity of the CRCC is not fixed, but is dependant on the number of characters in the data record.

2. Because the parity of the CRCC is not fixed neither the LRC or the CRCC need necessarily be written, as if the CRCC has even parity then so will the LRC. The polynomial chosen for the CRC, however, guarantees that at least one of the check characters will always be written.

3. It is possible to correct, as well as detect, certain errors. For this to be possible certain conditions must be met, notably that all errors must be confined to a single track. In practice due to the severe limitations on error correction using this system (correction can only be guaranteed for single bit errors) error correction is not

often implemented. For more information on this subject see reference [3].

2.2.5.4. EOF mark format. The file mark consists of only two characters, a single character and its LRC. The spacing is four character spaces for 7-track and eight for 9-track. Note that for 9-track no CRCC is written as would be the case for a data record. Note also that as only one character is written the LRC is identical to this character.

2.3. Physical Interface

The tape transports are of industry standard design. Space does not allow all specifications to be described here, but the most important features of the interface will be described. Further information may be found in references [6] and [7].

2.3.1. All interface lines are TTL/DTL compatible and make use of negative logic (a low represents the logical true condition). Outputs are open collector and lines are terminated at the receiving end by conventional 220/330 Ohm terminations. In order to allow data transmission over a distance of several meters each line also has associated with it a ground line, allowing the interconnection to be made in the form of a twisted pair.

2.3.2. The interface consists of three connectors :

2.3.2.1. Read data (RDO-RD7, RDP). This connector has ten lines, eight data, one parity and one read strobe. Data from the tape transport is latched on the trailing

edge of this pulse by the tape controller. All lines on this connector are inputs to the controller.

2.3.2.2. Write data. This connector consists of twelve lines, all outputs from the controller :

2.3.2.2.1. Data lines (WDO-WD7, WDP). There are eight data lines (six for seven-track units) and a parity line.

2.3.2.2.2. Write data strobe (WDS). Data and parity are written on leading edge of this strobe. It has a minimum width of 1 μ S and a maximum width of 3 μ S. Data must be valid for at least 0.5 μ S before and after the leading edge of this strobe.

2.3.2.2.3. Write amplifier reset (WARS). This strobe automatically writes the LRC described above. Note that writing the LRC resets the transports NRZl encoding circuitry. This strobe is identical to the write data strobe except that the data on the transport's data lines is ignored, and the internally generated LRC is written.

2.3.2.2.4. Read threshold (RTH). This line selects a high read threshold when true. This is used only for read-after-write data checks.

2.3.2.3. Control. This connector has both input and outputs. The number of lines varies from manufacturer to manufacturer. The following outputs from the controller are common to all transports :

2.3.2.3.1. Select (SLT). When this line is true all drivers and receivers in the transport are enabled. The use of this line allows several transports to be connected to a common bus. Note that this line cannot be commoned, and must be wired separately to each tape transport.

2.3.2.3.2. Synchronous forward command (SFC). When this line goes true, and the tape transport is ready and on line, the tape ramps linearly up to its rated speed in the forward direction. When this line goes false the tape speed ramps down to zero.

2.3.2.3.3. Synchronous reverse command (SRC). This is the same as the synchronous forward command except that the tape moves backward.

2.3.2.3.4. Rewind (RWD). A pulse on this line (minimum width 20 μ S) will cause the tape to rewind to the BOT marker.

2.3.2.3.5. Set write status (SWS). For the transport's write head to be enabled this line must be active at the leading edge of the SFC command, and must remain true for at least 10 μ S. Note that as magnetic tape is a sequential medium, reverse write operations are not used.

2.3.2.3.6. Off line command (OFFC). A pulse on this line will set the selected transport off line. This is normally done to indicate that the tape may be removed. The transport can only be put

back on line by an operator using the transport's front panel controls. The pulse must be at least 2 μ S in length.

In addition to requiring the inputs described above all transports provide certain status outputs :

2.3.2.3.7. Transport ready (RDY). This line is true if the transport is on line and a tape is loaded and is not rewinding.

2.3.2.3.8. On line (ONL). This line is true if the transport is on line. If the transport is on line it is under the control of the controller, if not it is under local front panel control.

2.3.2.3.9. File protect (FPT). This line is true if a write enable ring is not mounted on the tape loaded on the transport.

2.3.2.3.10. Load point (LDP). This line is true if the tape is at BOT.

2.3.2.3.11. End of tape (EOT). This line is true if the tape is past the EOT marker.

2.3.2.3.12. Rewinding (RWD). This line is true if the transport is engaged in a rewind operation.

2.3.3. The following tape transports are in use :

2.3.3.1. Pertec model 6860-25. This is a simple tape transport which operates at 25 IPS and conforms closely to the standard described above. It does however have the ability to overwrite a specific block. This ability is not in fact used in this application, as very few other tape transports (or computers systems) make provision for such a capability.

2.3.3.2. WANGCO Mod 10. This tape drive differs in several respects from the Pertec unit described above.

2.3.3.2.1. It operates at 45 IPS.

2.3.3.2.2. The interface has been modified slightly from the industry standard described above :

1. Provision has been made for four select lines in order to allow several transports to be bussed together without requiring special wiring for the select line.

2. Provision has been made for DC power connections (designed to power external terminator modules) on certain lines previously used for ground connections.

2.3.3.2.3. The transport has a dual gap head. This means that the unit is capable of reading and writing simultaneously. As the read gap is downstream of the write gap a tape can be written and verified in one pass, provided that the controller supports this feature.

2.4. Tape transport timing

It is the responsibility of the controller to generate the timing of the control signals in such a way as to produce the data format described above. These timing requirements will now be discussed. Note that 45 IPS is the maximum speed available in this type of tape transport. A controller capable of operating at this speed will be compatible with all industry standard tape transports.

2.4.1. Character timing.

2.4.1.1. Write operations. When writing data to the tape transport the bit density is controlled by the speed of the tape and the frequency of the write strobes. The controller must generate write strobes at the frequency required to produce the bit density appropriate for the tape transport in use. In the case of the transports discussed above, this is 800 BPI. This means that a write strobe must be generated each 50 μ S for the 25 IPS unit and each 27.78 μ S for the 45 IPS unit. In order to conform to the IBM specification an accuracy of 1% is required. Note also that this means that the controller must be capable of delivering nine bits of data each 27.78 μ S.

2.4.1.2. Read operations. During read operations data from the tape transport is latched by the read strobe. The average period of this strobe will be the same as the write strobe discussed above but due to skew between channels and bit crowding effects the minimum time between consecutive read strobes may be considerably shorter than this. Most manufactures state that a guaranteed safe value is half of the average

time between strobes. This means that when reading the controller must be able to process nine bits of data in less than 13.89 μ S.

2.4.2. Block timing. There are two aspects to the timing involved in reading blocks :

2.4.2.1. Gaps inside blocks. The check characters at the end of a block are delimited by a space of several bit intervals. The controller must be able to generate these gaps when writing. In order to be able to recognize the check characters, these gaps must also be detected by the read circuits of the timing module.

2.4.2.2. Inter block gaps. Gaps must also be inserted in between blocks. This is done by controlling the delay between asserting the motion command (SFC) and the first write strobe, as well as the delay between the last strobe and deactivating the motion command. These delays vary from approximately 2 mS to 100 mS for transports ranging in speed from 10 IPS to 45 IPS. These gaps are not critical, but should be controlled to within 5%.

2.5. Hardware requirements

2.5.1. The tape controller consists of a CPU card, memory card and timing module. From the description given above several requirements for the timing module hardware become apparent. The most significant are summarized below :

2.5.1.1. A character clock variable from 2 kHz (10 IPS, 200 BPI) to 36 kHz (45 IPS, 800 BPI) must be generated to within 1% accuracy.

2.5.1.2. Variable Inter record gaps must be generated. These range from 2 mS to 100 mS.

2.5.1.3. Gaps within blocks must be recognized. These range from 100 uS (4 character spaces, 800 BPI, 45 IPS) to 4 mS (8 character spaces, 200 BPI, 10 IPS).

2.5.1.4. The controller must be capable of processing nine bits of data within 13.89 uS.

2.5.2. Of the above requirements it is the last that is by far the most demanding. The first three requirements may be met simply by the use of programmable timers and the appropriate logic, but in order to meet the last requirement a controller architecture is required which is capable of reading 8 Kbytes of data plus its parity into some form of memory. In order to meet this requirement several techniques are in common use [24], [25], [26] :

2.5.2.1. Micro-programmed controller. Most commercial tape controller modules are constructed using bit-slice devices. These devices may be viewed as building blocks for central processing units, and allow the user to effectively create a computer with an architecture and instruction set optimized for the particular application in question. In this case the "computer" would have an instruction set composed of instructions such as read block and write block, and would be "programmed" by the host processor in the controller. These building block devices are known as bit slices because

they are normally supplied in the form of 4 bit "slices" which may be paralleled to achieve the required data width. A timing module would typically consist of three devices in parallel, giving a 12 bit word. This form of construction has several significant advantages :

2.5.2.1.1. As these devices operate at very high clock rates (30 MHz devices are not uncommon [27]) and may be optimized for the required function a controller designed in this way can normally perform all the functions required (timing, check character generation etc.) with both a minimum of hardware and a minimum of host processor intervention. A timing module based on this technology can easily meet the data transfer rate requirement as data may be easily read into local memory by program transfer (or, more accurately, microprogram transfer).

2.5.2.1.2. Modifications may be made reasonably easily as only a change to a PROM (Programmable Read Only Memory) will normally be required.

2.5.2.1.3. As all functions are performed by the bit slices and a few support chips the system can be made physically small (typically two SABUS cards).

This system however also has severe disadvantages in this particular application :

2.5.2.1.4. Development work is expensive. Changes require new PROMs, as these cannot be reprogrammed as can the EPROMs used for microprocessor work.

2.5.2.1.5. UCT does not have development equipment suitable for the design of such equipment.

2.5.2.2. A design composed of discrete logic elements in which data is transferred to and from a local memory. In this type of controller all functions, including the generation of the check characters, are done in hardware. This is the classic design for a tape transport controller and indeed the interface to the tape controller is optimized for such a system. This technique has the advantages that no problems occur with transfer speed as all operation are done in hardware, and that no special purpose development equipment is required. This technique does however also suffer from certain disadvantages :

2.5.2.2.1. Such a module would be very large. A preliminary estimate was that six SABUS cards would be required.

2.5.2.2.2. The system would not be very versatile. As all the required functions would be implemented in hardware it would be very difficult to modify the controller. For example, if error correction was desired as discussed in section 2.2.5.3.2. above, this would have to be designed in from the beginning. Further complexity would arise as regards the possible use of seven-track tape transports as, for example, these units do not make use of a CRCC.

2.5.2.2.3. A controller based on this type of technology would be very inefficient. Certain operations, such as the generation of check characters, are much better carried out in software, at a considerable saving in hardware.

2.5.2.3. The third possible controller design is a system in which the minimum of hardware is used, and a CPU within the controller performs as much as possible of each function in software. The hardware performs only those parts of functions requiring precision timing, and provides latches and buffering of the signals to and from the tape transports. This has the following advantages :

2.5.2.3.1. The timing module is physically small.

2.5.2.3.2. As most of each operations is carried out by the software the controller will be very versatile. For example error correction can be included without the addition of any extra hardware, simply by writing suitable software.

2.5.2.3.3. It is possible to provide more sophisticated diagnostics functions, as the CPU has direct access to most of the timing module's functions.

This organization, however, also has a disadvantage. In order to meet the data transfer rate requirement without memory which is local to the timing module, DMA (Direct Memory Access) techniques must be used. This is significant problem as SABUS based systems have in the past normally made use only of programmed transfer techniques (that is, transfers under the direct control of the CPU) to transfer data. The occasional system which has made use of DMA has been built as a dedicated system, and as a result no general technique for the use of DMA on SABUS has been developed. This disadvantage is however overshadowed by the advantages of size and simplicity to be gained. The following chapter will discuss the use of DMA on the SABUS system.

CHAPTER 3

THE 80188 CPU CARD

3.1. Introduction

The simplest way to design the the tape controller system is to utilize DMA. This is somewhat problematic, since DMA has not yet been satisfactorily implemented on SABUS. This chapter describes the implementation of a central processor card which implements DMA.

3.2. SABUS

SABUS is a bus system designed for the implementation of 8- and 16-bit microcomputer systems. The standard was first formulated in 1978, but has been modified considerably since. For reasons of space a only a brief description of the bus is included here. Further information may be obtained from references [8] and [9]. The bus has the following features.

3.2.1. All lines are TTL compatible, and 5 Vdc, 12 Vdc and -12 Vdc power supply lines are included.

3.2.2. A 20 bit address bus (AB0-AB19).

3.2.3. Either an 8- or 16-bit data bus (DB0-DB15). Also included is a BHE (Bus High Enable) line as used on Intel 16-bit CPUs. This allows the CPU to execute 8 bit operations on a 16 bit memory device.

3.2.4. Conventional memory read (MR), memory write (MW), IO read (IR) and IO write (OW) strobes.

3.2.5. Reset line (RESET). When this line is asserted all devices on the bus are reset.

3.2.6. Wait line (WAIT). When this line is asserted, the CPU card lengthens the active memory or IO access. This allows the use of memory and IO devices which are slower than the CPU card in use.

3.2.7. Clock (CLK). This line is driven by the CPU clock.

3.2.8. Hold (HOLD) and hold acknowledge (HOLDA). When hold is asserted by a peripheral card, the CPU card deactivates all its address, data and control lines, and then asserts HOLDA. This allows the peripheral card to control the bus. Note that not all cards support this line.

Various other lines also exist, notably a set of lines which are intended to support both multiple daisy-chained interrupts and multiple CPU cards. This has however never been implemented. It

should be noted that the provisions of the SABUS "standard" have on several occasions been modified by the control committee, and have also been "improved" or simply ignored by several manufacturers. The result is a system in which the only the power lines, address lines, data lines and read/write strobes are truly standard as regards function. At no time have any generally accepted timing specifications been published for SABUS.

3.3. Implementation of direct memory access

In light of the information given above, several possible approaches to the problem of introducing DMA suggest themselves :

3.3.1. Provide some form of DMA on the timing module, for example an 8237 DMA controller chip, or a special purpose circuit constructed from discrete logic components. The CPU card's execution would be suspended by the asserting the HOLD line, and the address, data and control lines would be driven by the timing module. This approach has the advantage that it is conceptually simple and allows the use of any CPU card which supports the HOLD/HOLDA lines. There are also certain disadvantages :

3.3.1.1. SABUS cards are not standardized as regards their use of the SABUS HOLD and HOLDA lines. These lines are on the bus specifically to allow DMA operations, but details of their operation and timing have never been formalized. Indeed the only published data[8] on the use of DMA on SABUS defines these lines in such a way as to make them incompatible with any chip other than the Intel 8257 DMA controller chip. This device is now obsolete. It is also too slow for

this application, and is incompatible with any CPU card in use at UCT.

3.3.1.2. The local area network card mentioned in the introduction requires the HOLD and HOLDA lines. As SABUS does not make provision for the use of these lines by more than one card, the use of these lines would make it impossible to include the LAN card into the tape controller at a later date. As discussed earlier, this future capability is a major design goal.

3.3.1.3. If the DMA components were to be included on the timing module this would increase its size considerably. For example buffers to drive all the SABUS address lines would have to be included.

3.3.2. An existing 8088/8086 card with DMA capability can be used. The card in question has provision for the inclusion of an Intel 8089 coprocessor. This is a device, designed specifically for the 8088/8086 family of microprocessors, which is designed to unload the CPU of as much IO activity as possible. The 8089's instructions are optimized for IO activity, and the device includes two DMA channels. The DMA request and acknowledge lines are brought out on a connector on the rear of the card as there is no provision for them on the bus. The advantage of this approach is that it is an existing card that has DMA capability. Several severe problems, however, exist :

3.3.2.1. The card has never been tested in conjunction with the 8089 coprocessor.

3.3.2.2. The card does not implement the SABUS HOLD/HOLDA lines, and so cannot be used with the LAN card.

3.3.2.3. No 8089 assembler is available at UCT.

3.3.2.4 The 8089 is relatively expensive (approx. R85) and is not held in stock in this country due to low demand.

3.3.3. A CPU card with built in DMA can be designed. This has several advantages.

3.3.3.1.1. The card can be designed in such a way that DMA channels are provided on card, and the card supports the HOLD/HOLDA protocol. This means that the card will support the UCT LAN card.

3.3.3.1.2. Provision can be made for interrupts. As discussed earlier communication to the host computer will be via a serial line. Although not vital, it would be both far simpler and far more efficient if this serial port were to be interrupt driven. No current SABUS card makes provision for this.

3.3.3.1.3. Provision could possibly be made for having EPROM on the CPU card. In the past SABUS systems have always had separate cards for EPROM storage. In recent years, however, the density of EPROMs has increased to the point that very few applications require more than one or two. A significant saving in in both size and expense could be realized if the EPROM storage could be integrated onto the CPU card.

3.3.3.2. As a result of the advantages described above it was decided to design and construct a CPU card which would have as many of the features listed above as possible. For this purpose the microprocessor chosen was the Intel 80188. It was chosen for the following reasons :

3.3.3.2.1. It is an improved version of the popular 8088/8086 range of microprocessors. UCT has available both an assembler and high level language for this series of processors. The availability of a high level language for this microprocessor is a considerable advantage. This eases the task of writing the software, and makes it easy to modify at a later date.

3.3.3.2.2. The device has, integrated on chip, peripheral and memory select logic. This means that any peripheral or memory devices (such as EPROMs) included on the CPU card will require no additional address decoding circuitry.

3.3.3.2.3. A two channel DMA controller is integrated on chip.

3.3.3.2.4. Also integrated on chip is a programmable wait state generator. This can be programmed to insert wait states into any reference to a memory or IO port location. This is an important consideration as it allows the microprocessor to interface to a wide variety of devices which do not have a sufficiently fast response time to interface without slowing the CPU down in some way. As SABUS has no formal timing specifications, a CPU which has programmable timing is a great advantage. Most other CPUs would require complicated hardware to insert wait states. This also

makes for great versatility. Almost any device can be added to the system without requiring changes to the hardware. As a further advantage, these programmed wait states also apply to the DMA controller. In systems which have a separate DMA controller the different timing requirements of the CPU and DMA controller can cause severe problems.

3.3.3.2.5. The microprocessor also has integrated on chip an interrupt controller which can be used to control the data link as discussed above. The device also contains programmable timers, which are useful for providing a real time clock.

3.4. The 80188 Microprocessor

For the reasons described above it was decided to design an 80188 based CPU card. Before the design of this card is discussed a brief description of the microprocessor will be given. This cannot, for reasons of space, be a full description, but further information may be found in references [10], [11] and [12]. The device contains the following sections :

3.4.1. Clock Generator. The clock generator requires only the addition of a crystal of twice the desired clock frequency. Also included in this section are reset and wait state generation circuitry.

3.4.2. Execution unit. The instruction set of the device is compatible with the 8088/8086 microprocessors. Certain extra instructions have been added, and most old instructions

execute significantly faster (typically 2 to 4 times as fast).

3.4.3. Interrupt control unit. This section synchronizes interrupt requests, prioritizes them and then passes control to the appropriate section of code. The interrupt controller has several modes of operation of which only the simplest is discussed here. Interrupts from several sources are recognized. All these interrupt sources, except for the non maskable interrupt, can be disabled and have a programmable priority.

3.4.3.1. Non maskable interrupt. This is an external interrupt, and cannot be disabled. It always has the highest priority of all the interrupts.

3.4.3.2. External interrupts. These are four interrupt lines which are brought out on pins of the 80188. They can be programmed for level or edge operation as well as priority and are driven by external peripherals.

3.4.3.3. DMA channel interrupts. Each DMA channel has an interrupt which occurs on terminal count.

3.4.3.4. Timer interrupt. Each integrated timer can be programmed to generate an interrupt on reaching its maximum count value.

3.4.4. Programmable timers. The microprocessor contains three programmable timers of two different types :

3.4.4.1. Two of the timers (TMR0, TMR1) have external input and output pins, and may be programmed to take

their input signal either from these pins or from the chip's internal clock. The high time and low time of the output signal may be independently varied.

3.4.4.2. The third timer (TMR2) has no external connections, and always takes its input from the CPU clock. In common with timers 0 and 1 it can generate interrupts, but has the additional ability to initiate DMA cycles.

3.4.5. DMA unit. The DMA controller consists of two identical DMA channels. Each channel can be programmed to transfer from an IO port to memory, memory to IO port or memory to memory. DMA operations may be either unsynchronized, or synchronized by the source or synchronized by the destination. In this particular application only source synchronization (for reading tapes) and destination synchronization (for writing tapes) are used. The use of these DMA channels is critical to the design of the tape controller, and will be further discussed in the following chapter, which discusses the design of the timing module.

3.4.6. Chip select unit. This unit provides chip selects for various peripheral and memory devices :

3.4.6.1. Peripheral chip selects (PCS0-PCS6). These are 7 chip select lines intended to select IO devices such as serial ports. The seven chip selects are active for seven contiguous areas of 128 bytes in either the processor's memory or I/O space. The number of wait states inserted into read or write cycles is programmable.

3.4.6.2. Memory chip selects. Three sets of memory select lines exist :

3.4.6.2.1. Upper chip select (UCS). This line selects a block of memory up to 256K bytes long. This block always ends at location 0FFFFFFH. This is the only select line which is active at reset. This section of memory must contain the boot code for the processor.

3.4.6.2.2. Lower chip select (LCS). This line selects a block of memory up to 256K bytes long. This block always starts at location 0H. It is almost always RAM (Random Access Memory).

3.4.6.2.4. Mid-range chip selects (MCS0-MCS3). These are four chip select lines of up to 128K bytes in length which form a contiguous block. The start address of this block is programmable.

All of these lines may be programmed for the number of wait states inserted, as well as whether the external ready signal should be taken into account.

3.4.7. Bus interface unit. This section of the processor generates the various control signals used by the system. These include read/write strobes, status lines, address and data lines as well as provision to halt the CPU (HOLD/HOLDA).

3.5. The CPU card design

From the discussion above it may be seen that several features are desirable in the CPU card :

3.5.1.1. Local EPROM storage.

3.5.1.2. Provision for interrupt driven serial I/O. As no existing serial interface cards make provision for this, it was decided to place two serial ports on the CPU card. This has the following advantages :

3.5.1.2.1. The USART (Universal Synchronous/Asynchronous Receiver/Transmitter) chip is connected directly to the 80188 external interrupt lines. Any other organization would require an extra connector, as SABUS does not make adequate provision for interrupt lines.

3.5.1.2.2. The USART chip selects are driven from the 80188's PCS lines. This saves address decoding circuitry.

3.5.1.2.3. The USARTs' baud rates are generated by the 80188's internal timers. This saves space and gives the system two serial ports with independently programmable baud rates.

3.5.2. As a result of the above considerations, and as a result of the card size, the design shown in figure 3.1. was decided on. This consists of the following sections :

3.5.2.1. 80188 CPU, with a local and master bus. The master bus is SABUS, and the USARTs and EPROMs reside on the local bus. This saves on data bus buffers for these devices, but also means that they cannot be accessed from SABUS. It is however unlikely that this will be required. An Intel 8288 bus controller was chosen to control the master bus. Connection to the CPU's two DMA request lines, one for each DMA channel, are made via a connector on the rear of the card, as no provision is made for such lines on the SABUS connector.

3.5.2.2. Two 8251A USARTs and voltage level conversion devices for RS-232 compatibility. Connection to these two serial ports also is made via a connector on the rear of the CPU card, as is the case for the DMA request lines.

3.5.2.3. Four 28 pin byte-wide device sockets. This allows any available EPROM to be inserted, with only software modification. These devices are selected by the 80188's memory device chip selects. This allows up to 128K bytes of memory using currently available devices.

3.5.2.4. SABUS interface. The SABUS interface performs the following functions :

3.5.2.4.1. Buffering. Buffers are provided for the address, data and control lines. As discussed in section 3.5.2.1., SABUS is CPU's master bus. Note the buffering for the SABUS control lines is provided by the 8288 discussed in section 3.5.2.1. above, which also generates these signals.

3.5.2.4.2. Ready line synchronization. The 80188 CPU requires that transitions on its ready line (used to extend bus cycles for memory or IO devices for which a normal bus cycle is too fast) are synchronous with the CPU clock. Circuitry is provided to do this. This circuitry also lengthens the first bus bus cycle after a hold condition is releases, in order to ensure that the 80188 CPU's timing requirements are met.

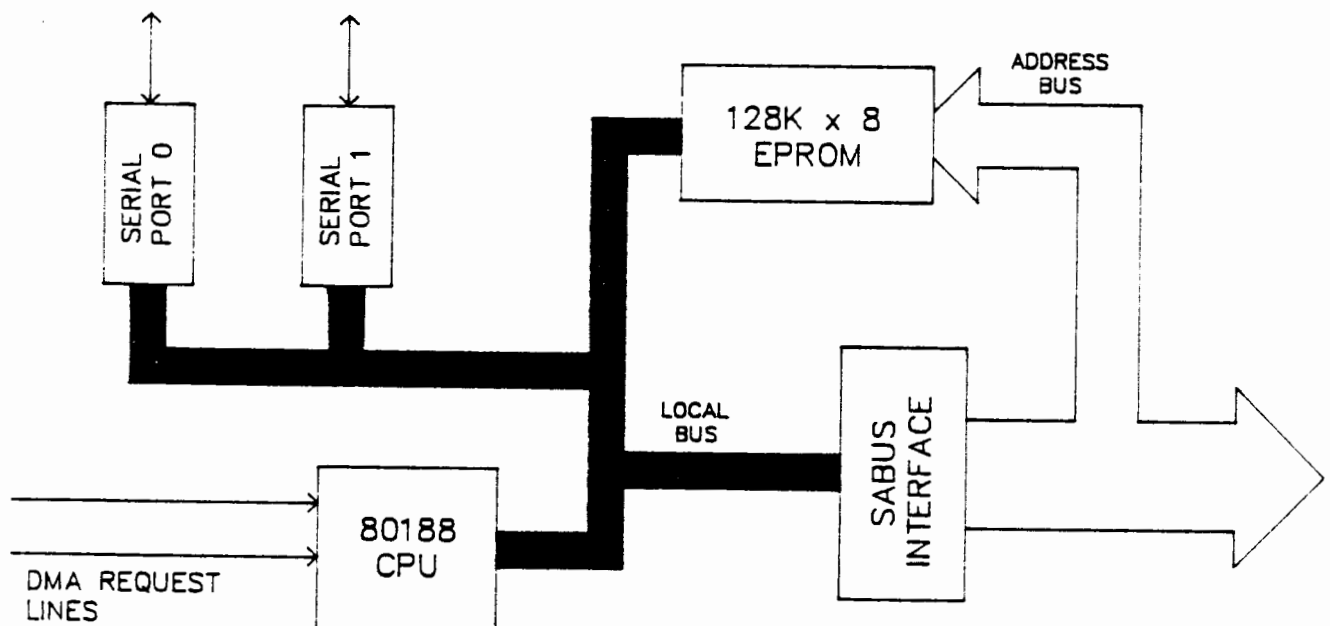


Figure 3.1.

3.5.2.4.3. Clock line drive. The CPU clock is divided down, and used to drive the SABUS clock line.

3.5.2.4.4. Reset line drive. When the CPU card is reset, or is powered up, the SABUS reset line is pulled active by circuitry on the CPU card. This ensures that all peripheral devices on SABUS are reset.

3.6. Detailed design description

A detailed description and schematic diagram of the CPU card are provided in appendix B.

3.7. Testing the CPU card

Testing of the CPU card was done in several steps :

3.7.1. Initial testing involved writing a program, resident in the boot EPROM, which simply executed a continuous loop. This allowed the verification of the basic operation of the hardware, such as the clock and addressing of the boot EPROM.

3.7.2. The next step in testing involved writing a routine to output a fixed character string to serial port 0. This verified both the operation of the serial port and the CPU's on chip timers.

3.7.3. Once the operation of the serial ports had been verified a program which required the use of RAM was written. This was first tested with a static RAM card which does not make use of the SABUS WAIT line, in order to test the basic functioning of the SABUS interface. Once this was verified the static RAM card was replaced by the dynamic RAM card to be used in the final product. Use of this card verified the correct operation of the wait synchronization logic.

3.7.4. The final step in the process was to write a hex down load program. This program accepts an Intel format hex file on serial port 0, loads this into RAM and then executes it. This program was used to down load all later software, so avoiding the need to continually program, erase and reprogram EPROMs.

CHAPTER 4

THE TAPE INTERFACE MODULE

4.1. Introduction

The tape interface module consists of a pair of SABUS sized card which provide the physical interface between the controller and the tape transports, as well as certain timing and control functions. In the previous chapters the basic principles on which the design of the module is based were discussed. The module consists only of enough hardware to provide the bare minimum of functions, and makes use of DMA provided by the CPU card to execute high speed data transfers. This chapter provides a detailed description of the requirements and design of this module.

4.2. Requirements

The requirements of the module are as follows :

4.2.1. Electrical compatibility with the tape transports. The specifications of the transports were described in detail in a previous chapter, but may be summarized as follows :

4.2.1.1. All outputs to the tape transport are open collector, high power TTL lines.

4.2.1.2. All inputs to the interface module must be terminated by 330/220 Ohm TTL terminations.

4.2.2. The interface module should provide all critical timing. When designing microprocessor based equipment there is always a temptation to use the microprocessor itself to provide timing. This has the advantage of saving hardware, but has certain disadvantages both in general and in this specific case :

4.2.2.1. In general, use of a program to control timing makes the program difficult to modify and difficult to document.

4.2.2.2. The time taken to execute a section of a program will not be constant. This is for the following reasons :

4.2.2.2.1. Interrupts. The data link is interrupt driven. Any interrupt activity makes the program timing unpredictable. If the microprocessor is providing the timing, then such activity cannot be allowed while a tape operation is in progress. The software could be so designed to ensure that this is the case, but this is a restriction which is

best avoided, especially as it would introduce undesirable interaction between the data communications and the tape operations sections of the program.

4.2.2.2.2. DMA activity. The DMA controller acts as a cycle stealing device. This means that the program may be suspended at any time to allow DMA activity. It may be expected that there will be DMA activity exactly at the time at which the timing of the tape operation is the most critical.

4.2.3. A study of the abilities required from the tape controller by the PDP-11, and the abilities of commercially available tape controllers shows that the controller should be able to execute the following set of operations :

4.2.3.1. Read block. The controller must be able to read blocks both forward and reverse.

4.2.3.2. Write block. Block writes must be done only in the forward direction, and the inter-block gaps must be programmable. The controller should produce gaps of the same length regardless of what the last operation was (for example, read or write).

4.2.3.3. Write tape mark. The same conditions as for writing a block apply.

4.2.3.4. Space blocks. The controller should be able to space blocks either forward or reverse. Block space operations are always aborted on encountering a tape mark.

4.2.3.5. Space files. This operation is required by the PDP-11 operating system. It is not however generally implemented in hardware as files are delimited in different ways depending on tape format. For example in the case of simple 'unformatted' (no format information in data records) tapes, files are delimited by tape marks, whereas in the case of ANSI format tapes, files are delimited by special records. As a result, the control of file space operations is left to software.

4.2.3.6. Rewind. The unit must, of course, allow rewind operations. It should be noted that the controller may, unlike for all other tape operations, assume the success of this operation. This is done as rewind operations can take a considerable amount of time, and no error condition other than gross transport failure can occur. As a result the controller should be able to sense when the transport is executing a rewind, and delay any subsequent operations until this is complete.

4.2.3.7. Off-line. The unit is set off-line when the tape must be changed, or other operator action is called for.

4.2.3.8. Status read. The status of a particular tape drive may be read.

4.2.4. Various other capabilities are desirable :

4.2.4.1. The controller should be efficient. By this is meant that a minimum of time should be taken for the various operations. For example when spacing blocks the

tape should not have to stop at each block, nor indeed should any action on the part of the CPU be necessary.

4.2.4.2. The controller should be software intensive. As many as possible of the various operations should be performed by varying the software rather than the hardware. For example a single general hardware write operation should be capable of writing either a data record or a tape mark, simply by varying the programming of the controller hardware.

4.3. Design of the tape interface module

The various operations to be performed may be divided into four sections :

4.3.1. Status operations. These are operations which require little or no timing. These include status reads, rewind and off-line operations. The only hardware required to support are parallel I/O lines and buffers. The timing required is limited to providing minimum pulse widths on rewind and off-line operations, which can easily be done in software because the maximum pulse width is not critical.

4.3.2. Write operations. It may be seen from the diagrams in chapter 2 that the various write operations can all be performed as a single general write. A write with long gap only requires modified timing, and a write tape mark is simply a normal write block operation of only one character and a CRCC of all zeros. Several aspects of the timing of the write operation must be accurately controlled :

4.3.2.1. Character clock. This is the basic rate at which characters are written to the tape transport, and must be generated by the tape interface module. It is primarily used to generate write strobes, but is also used as the basic reference to set gap lengths, as will be explained later. The frequency of this clock is proportional to the tape speed, and must be accurate to 1%. From the discussion in chapter 2 it can be seen that it is necessary to change the gap between write strobes for the check characters. This may be achieved either by changing the division ratio of the character clock generator, or simply disabling the strobe for a number of character clocks. Of these two techniques the second is to be preferred, as it may be achieved with a minimal amount of extra hardware.

The design calls for nine bits of data to be transferred for each character written, but the DMA system transfers in bytes (eight bits). As a result, two full bytes must be transferred for each character written. This means that seven extra bits are available for control information. Two of these bits may be used as strobe enables, one to enable write strobes, and one to enable the WARS (write amplifier reset) strobe. The use of DMA will be further described in a later section.

Operation of this system is simple. In the case of nine-track operation the sequence of events is as follows :

4.3.2.1.1. The data characters are transferred to the interface module, all with the write strobe enable bit set, so generating write strobes.

4.3.2.1.2. Three dummy characters are transferred, with no strobe enable bits set. This generates a gap of four character spaces as required.

4.3.2.1.3. The CRCC is transferred with the write strobe enable bit set. This writes the CRCC.

4.3.2.1.4. Three further dummy characters are transferred as described above, generating the second required gap.

4.3.2.1.5. Finally another dummy character is transferred, but with it's WARS enable bit set. This results in the LRC being written.

4.3.2.2. Inter-block gaps. In order to write tapes with the correct inter-block gaps the following time periods must be controlled :

4.3.2.2.1. Time delay between starting tape motion (with SFC) and writing the first character. This, in conjunction with the position of the head as a result of the previous operation, sets the inter-block gap.

4.3.2.2.2. Time delay between writing the LRC and deactivating the motion command. This sets the point at which the head will stop.

4.3.2.2.3. Also required, for obvious reasons, is a counter for the number of characters to be written.

These times can be set by means of three counters in sequence, the first counting the start gap, the second the number of characters, and the third the stop gap. The second counter must clearly be clocked by the character clock, but in fact it is advantageous to have all three counters clocked at this rate. The reason for this is that most manufacturers of tape transports set the start and stop times in such a way as to make them inversely proportional to the tape velocity. This means that if the counters which set the gaps are clocked from the character clock (proportional to tape velocity) the counter setting remain constant, regardless of tape velocity.

4.3.3. Read operations. Read block operations are in general simply a matter of setting the tape in motion and waiting for read strobes. As each character is clocked in it is transferred to memory by DMA. The operation of the DMA circuitry will be further described in a later section. There are however three aspects of the timing of read operation that deserve attention :

4.3.3.1. It is important that the position of the head after a read operation is known precisely, so that a possible subsequent write operation will have the correct inter-block gap. This means that a counter is required to set the delay from the end of the block to removing the tape motion command.

4.3.3.2. From the above it can also be seen that it is important to be able to detect the end of a block accurately and repeatably.

4.3.3.3. The controller should also have some way of distinguishing the check characters from the data in a block. This is not absolutely required for simply reading a block and detecting errors, but is convenient, and is mandatory if error correction is contemplated.

All of the above requirements are inter-related, and will be discussed once the requirements of space operations have been described.

4.3.4. Space operations. Space operations are performed in hardware only for blocks. Space operations must stop on detecting a tape mark, and so it is necessary for the hardware to be able to detect a tape mark without CPU intervention. How this is accomplished will be discussed in the next section. It should be noted that space operations must also, as for read operations, stop with the head in a known position relative to the last block or tape mark passed.

4.4. Format recognition

4.4.1. In the sections above the requirements for the various tape operations were discussed. From these it is obvious that the timing module must be able to recognize certain elements of the tape format. These are as follows :

4.4.1.1. Inter-block gaps. The timing module must recognize inter-block gaps for two purposes :

4.4.1.1.1. The inter-block gap terminates a read block operation.

4.4.1.1.2. For space operations the blocks must be counted. This means that no glitches or double transitions can be tolerated, as these would be counted as blocks.

4.4.1.2. End of file marks. The timing module must detect tape marks both in order to inform the controlling program that a tape mark was passed, as well as in order to terminate a space blocks operation.

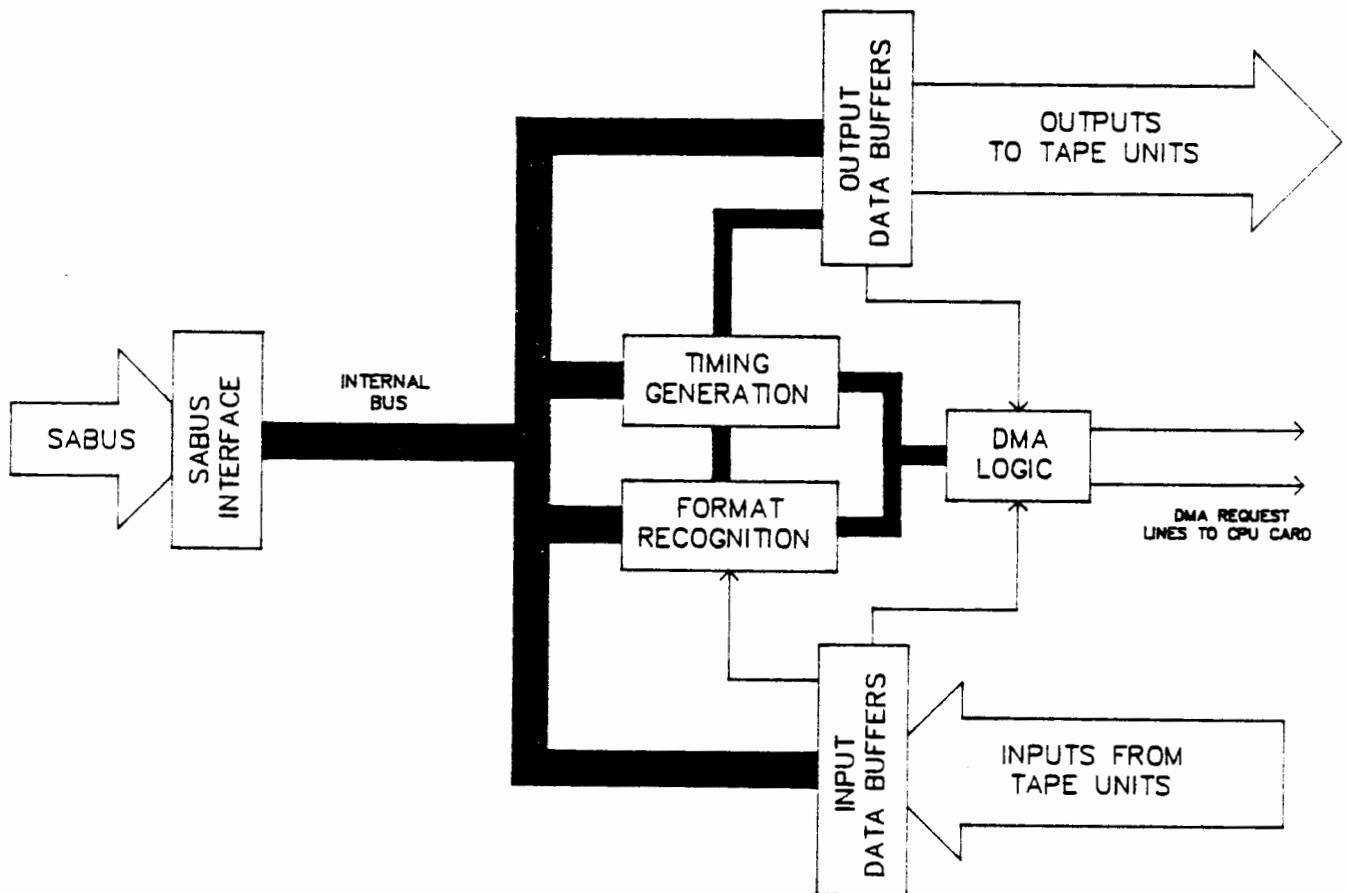


Figure 4.1.

4.4.1.3. Check characters. Check characters must be detected in order to allow error correction. Note that even if error detection is not contemplated, the availability of status information indicting whether or not a character is a check character not only simplifies programming, but also provides an extra check on data validity.

4.4.2. In order to meet these requirements a state recognizer was designed. This makes use of the timing signals from the tape transport to synchronously track at which point in a block the tape head is. This state information is read in together with each input character, so allowing the program to determine whether a particular character is a data or check character. In addition this state information, when decoded and processed with further timing information, provides signals denoting that the tape head has passed over a file mark and that the tape head is passing over an inter-block gap. A detailed description of this circuit is given in section 4.6.

4.5. DMA Logic.

4.5.1. In previous sections of this chapter, reference was made to the use of DMA for both write and read operations. In section 4.3.2.1. the need for to bytes of data to be transferred for write operations was discussed. Similarly for read operations a minimum of nine bits (eight data and one parity) must be transferred for each character read from tape. As the data is read/written from one port, and the parity and associated information read/written from another port it is necessary to use two DMA channels for both read and write operations.

DMA requests are generated by the tape interface card setting the DMA request lines active. Note that these lines do not form a part of SABUS, but are on an auxiliary connector on the back of

both the CPU and tape interface card. Connection between these two connectors is made by ribbon cable. It should further be noted that as the CPU does not generate DMA acknowledge signals these must be synthesized by the tape controller card from the read and write strobes to the IO ports in question. The operation of the DMA system will now be described individually for read and write operations.

4.5.1.1. Write operations. For write operations the DMA controller is set for destination synchronization. This means that each DMA channel will transfer a single byte from memory to an IO port on each occasion that the DMA channel in question's DMA request line is taken active. The destination address of DMA channel 0 is set to the address of the data output port of the tape interface module, while the destination address of DMA channel 1 is set to the address of the parity and strobe enable port. The source addresses of the DMA channels are set respectively to the address of a block of memory containing the data to be written, and to the address of a block of memory containing the parity and strobe information to be used. As soon as the operation is initiated, the two DMA request flip-flops in the tape interface module are set, taking the DMA request lines active. The first character, its parity and its strobe information is then transferred to the tape interface's output latches/buffers. Each write operation resets the appropriate DMA request flip-flop. The data stored in the latches is written to tape by a write strobe generated by the timing section. This strobe also sets both DMA request flip-flops, resulting in the transfer of the next character and its associated information. This process continues until the write operation is complete. Note that as the timing of the write strobes is set by the timing section, rather than the DMA system, the only timing specification which the DMA

system must meet is that of maximum time from DMA request line active to transfer completed.

4.5.1.2. Read operations. Operation of the DMA system for read operations is very similar to that for write operations, with certain exceptions. For read operations the DMA controller is set for source synchronization. This means that each DMA channel will read a single byte from an IO port into memory on each occasion that the DMA channel in question's DMA request line is taken active. The source address of DMA channel 0 is set to the address of the data input port of the tape interface module, while the source address of DMA channel 1 is set to the address of the parity input port. The destination addresses of the DMA channels are set respectively to the address of a block of memory into which the data must be read, and to the address of a block of memory into which the parity information must be read. Each read strobe sets two DMA request flip-flops, and the character read in and its parity information is then transferred to the tape controller memory. Each read operation resets the appropriate DMA request flip-flop. This process continues until the read operation is complete. As is the case for write operations, the only timing specification which the DMA system must meet is that of maximum time from DMA request line active to transfer completed. Note that the DMA channels remain active until the DMA operation is aborted by the program, once the end of the block being read has been reached.

4.6. Design Description

A block diagram of the timing module is shown in figure 4.1., and appendix C contains a detailed circuit diagram and circuit

description. The module is composed of two SABUS cards. These cards are connected via a 40 way ribbon cable of which two lines are also connected to the CPU card. These are the DMA request lines. Connection to SABUS is made only via the SABUS connector on card 1, with the connector on card 2 being used only as a mechanical connection. This allows both cards to be simultaneously debugged with only one extender card. Card 1 also has the write data and control connectors to the tape transports, while card 2 contains the read data connector. The design will now be described in terms of the blocks given in figure 4.1.

4.6.1. SABUS interface. The SABUS interface's function is to provide address decoding, as well as buffering for the data bus and control lines. The base address of the module is wire-wrap selectable. All of the components related to the SABUS interface are located on card 1.

4.6.2. Output buffers. The output buffer section of the module provides both buffering and latching of signal to be outputted to the tape transports. All of these devices reside on card 1. This section may be further subdivided into three subsections :

4.6.2.1. Data output latches. These latches latch the output data and parity information to the tape drives, as well as strobe enable information. The data and parity bits are buffered and transmitted to the tape transport, while the strobe enable bits are gated with signals from the timing generator to provide write and write-reset strobes to the transports. Data transfer to these latches is always by DMA, as described in section 4.5.

4.6.2.2. Configuration latch. This 8-bit latch sets the configuration of the module. It has the following outputs :

4.6.2.2.1. Drive select lines. Two bits are used to select the tape transport.

4.6.2.2.2. Count select lines. These two lines are fed to the timing section of the module, and will be described in a later section.

4.6.2.2.3. Forward/reverse select line. This line selects the direction of tape motion.

4.6.2.2.4. Read/write select line. This line selects whether a read or a write operation is to be performed.

4.6.2.3. Command output latch. It is the function of the command output latch to provide signals to the module which may be used to drive edge sensitive inputs. In order to achieve this a bit addressable latch is used, so as to allow only one bit at a time to be updated without raising the possibility of unwanted transitions on any other bit. This latch provides edge sensitive signals, such as the tape motion command lines, direct to the tape drive as well as a GO (start operation) and a board reset signal to the rest of the module. The outputs of the latch are reset to a logical zero by the SABUS reset signal, which sets the board reset active, and all other signals inactive. This prevents the module from generating any unwanted signal during power up.

4.6.3. Input buffers. The input buffer and latch section resides on card 2, and consists of two subsections :

4.6.3.1. Data input buffers and latches. These two 8-bit latches latch data and parity from the tape transports, as well as status information from the rest of the module. These status bits include state information from the format recognition section, and a bit from the DMA section which indicates the occurrence of a DMA error. This status information is stored for each character which is read in. The latches are clocked by the read strobe from the selected tape transport. Transfer from both these buffers is always by DMA, as is that of the latches used to output data and parity information.

4.6.3.2. Status input buffer. This buffer buffers several lines which are taken directly from the tape transports, as well as two signals from the timing module itself. These signal are a run signal, which indicates whether the module is executing a command, and an EOF signal, which indicates that a tape mark was passed during the process of the last command. Note that these signals are not latched, and represent the status of the unit at the time of the read, rather than at the time that the previous operation terminated, as is the case for the other input lines.

4.6.4. Format recognition section. The function of this section, which resides entirely on card 2, was described above. It consists essentially of a state machine, timing circuits and state decoding logic. The state machine consists of three JK flip-flops and their associated gates. The state machine is clocked by the read strobe from the

tape transports, and has two inputs which indicate timing information. These two input are SG (short gap) and LG (long gap). SG is active when a strobe has not been detected for two character clocks, and LG when a strobe has not been detected for ten character clocks. Both of these signal are generated by programmable timers, so allowing operation with tape transport of various speeds. The state decoding logic provides two signals, the first denoting that the head is passing over a inter-block gap, and the second that an EOF mark has been passed.

4.6.5. Timing generation section. This section generates signals which require precision timing, such as the tape motion signals and the write strobes. The timing section consists of several sections :

4.6.5.1. Master clock. The master clock consists of a Colpitts oscillator, which was chosen above logic gate oscillators for its superior accuracy and start-up performance [13]. This section generates a 5 MHz square wave, which is used to derive all other module timing.

4.6.5.2. Character clock generation. The character clock is generated by dividing the master clock by a programmable ratio. This signal is used to generate write strobes, as well as for timing purposes.

4.6.5.3. Start counter. This counter sets the delay from the start of an operation to the beginning of the count phase of the operation. This only has real significance in the case of write operations, when this is when the first character is written. For all other operations this counter, although still in the timing

chain, is set to a small nominal value. This counter is programmed in multiples of character clocks.

4.6.5.4. Main counter. This counter counts either write strobes, read strobes or blocks, as defined by the count select lines mentioned in section 4.6.2.2. The time during which this counter is active is known as the count phase, and this phase may be ended either by the counter reaching zero or by a programmable end condition. This end condition is programmed by the same count select lines as the items to be counted. Three combinations are valid for the count select lines. These are as follows :

4.6.5.4.1. Count select of 00. This value selects the counting of write strobes, and no end condition. This is used for write operations, which always end only when all characters have been written.

4.6.5.4.2. Count select of 01. This value selects the counting of read strobes, and an end condition of encountering a inter-record gap. This is used for read block operations, when the operation is normally terminated at the end of a block (e.i. when an inter-record gap is encountered) or when some maximum number of characters is exceeded. This last condition is an error, and the operation is ended to avoid buffer overflow.

4.6.5.4.3. Count select of 10. This value selects the counting of inter-record gaps encountered, and an end condition of passing a tape mark. This is used for space operations, where a specified

number of blocks must be spaced, and the operation must be aborted on encountering a tape mark.

4.6.5.5. Stop counter. Once the end of an operation is encountered (the last write strobe generated or read strobe encountered), a delay must occur before the tape motion command is deactivated, in order to ensure that the tape head stops at the correct point relative to the end of the last block or tape mark. This is accomplished by the stop counter, which is programmed in multiples of character clocks.

4.6.6. DMA request generation section. This section generates two DMA request signals which are passed to the CPU card. The operation of the DMA system was described extensively in section 4.5. This section consists essentially of six flip-flops. The first two are used to generate DMA requests for write operations. The first flip flop generates a request signal for the data latch's DMA channel, and the second for the parity and strobe enable latch's DMA channel. Both of these flip-flops are set by either a write strobe (including dummy writes) or by the GO signal. The GO signal loads the data for the the first write into the latches. The flip-flops are reset by a write to their respective latches. The next two flip-flops perform a similar function for read operations, with the exception that the flip-flops are set only by the read strobe. The final two flip-flops are used to detect DMA overrun errors which occur in the event of a DMA cycle being required before the previous cycle is complete.

4.6. Testing the timing module

Testing of the timing module proceeded in several steps :

4.6.1. As a first step the operation of the master oscillator was checked.

4.6.2. Following this the operation of the SABUS interface was checked. This involved ensuring that the module was being addressed at the correct base address, and that all ports were receiving strobes. For this purpose special programs were written.

4.6.3. The next step in the testing process was to check the timing generation section of the module. This step in fact proceed in tandem with the software development, simplified versions of the software being used to verify correct hardware operation. This process culminated in the verification (with a logic analyzer) that the timing module produced the correct waveforms for a write block operation. At this stage the operation of the DMA generation section was also verified.

4.6.4. Once the operation of the timing section of the module had been verified the operation of the format recognition circuit was tested. This was done simply by looping the interface back by connecting the WDS, WARS and RDS lines. In this way the correct operation of the format recognition circuit was checked on the logic analyzer.

4.6.5. Once the above were completed the module was tested in conjunction with an actual tape transport. First it was verified that the tape controller could read tapes that it

has written, and then it was verified that the controller could read tape written by other systems, and that the controller could write in a format that is readable by other systems.

CHAPTER 5

THE RSX-11M OPERATING SYSTEM

5.1. Introduction

RSX-11M is the operating system in use on UCT's PDP11-23 computer system. It is a multi-tasking, multi-user operating system, written and supported by Digital Equipment Corporation, the manufacturer of the PDP-11, and is designed to run on PDP-11 systems which support memory management. This chapter provides a very brief introduction to this operating system and it's I/O subsystem. More information may be found in references [16], [17] and [18].

As PDP-11 systems are used mainly in industry and scientific applications it is important that the operating system should be both fast and versatile enough to allow the user to introduce his own hardware into the system configuration. RSX-11M has been designed to provide a high level of performance in computer systems which have relatively small memory spaces, and limited processing power. As a result the operating system's design goals have been to minimize code size, and to maximize processing

speed. To meet these goals the RSX I/O system attempts to centralize as many as possible of the common I/O functions, thus eliminating repetitive code, as well as allowing this code to be written as efficiently as possible. In order to achieve this centralization, the system makes extensive use of data tables to drive the centralized code, reducing substantially the the amount of code required for individual devices.

5.2. I/O system structure

The I/O system is structured as a loose hierarchy. The term 'loose' here means that the hierarchy can be entered at any level, not just at the highest level as in a 'tight' hierarchy. Tight hierarchies exist primarily for reasons of security, but can be inefficient, and are not very versatile. The upper level of the hierarchy is split into two sections, privileged and non-privileged. Non-privileged tasks are normal user programs, while the privileged tasks are normally associated with the operating system itself, and cannot be executed by a normal user. Typical examples of privileged tasks include dismounting a disk or obtaining exclusive use of a peripheral device. For simplicity the discussion below will be limited to non-privileged tasks. These tasks issue I/O requests which are processed by the following layers of the operating system :

5.2.1. FCS/RMS. The File Control Services (FCS) and Record Management Services (RMS) are at the top of the I/O hierarchy. These services provide device independent access to I/O devices, in the form of file access and record access respectively. The FCS/RMS system functions essentially by converting high level (Get and Put) commands into QIOs (Queued I/O requests).

5.2.2. QIO directive. The QIO directive is the lowest level of I/O which a task can request. All QIO requests are processed by the executive to prevent tasks from interfering with each other.

5.2.3. Executive I/O processing. Once the QIO has been passed to the executive it is processed further by the following subsections of the executive :

5.2.3.1. An Ancillary Control Processor (ACP). An ACP is responsible for maintaining file structure and data integrity for devices which are file structured. The ACP translates virtual QIOs (the block to be processed is specified by position relative to the file which contains it) to logical QIOs (block specified as a physical block eg. sector 10, track 13 etc). An ACP typically exists for each class of file structured device in the system. It should be noted that the ACP is a task, rather than part of the executive itself.

5.2.3.2. Executive I/O processing. This can be divided into three areas :

5.2.3.2.1. QIO directive processing. This subsection of the executive provides services such as direction to the required device, multi-user protection etc.

5.2.3.2.2. The I/O driver (or device driver). The I/O driver provides the device dependant I/O code. Also associated with each device driver are data tables, which provide information to the executive relating to the status and abilities of the

device. Device drivers may be either resident (always in the system) or loadable (only inserted into the system if required). The driver's function will be discussed in detail in a later section.

5.2.3.2.3. I/O related subroutines. As discussed earlier, the executive provides a wide variety of services to the driver. These centralized subroutines provide services related to interrupt processing, decoding of I/O requests and the buffering of data. Several individual routines are discussed in a later section.

5.3. The role of the device driver

The device driver provides several services, which may be most easily described in terms of the driver's entry points.

5.3.1. I/O initiator. The executive calls this entry point to inform the driver that an I/O request is waiting to be serviced. The device driver then makes use of the appropriate executive routine to obtain information on what the desired operation is, and then executes the appropriate steps in hardware to complete the operation.

5.3.2. Cancel I/O. The executive calls the driver at this entry point if it becomes necessary to abort the current I/O request.

5.3.3. Time-out. The device driver can establish a time-out count. This is decremented by the executive at regular

intervals, and if it reaches zero a call to this entry point is executed.

5.3.4. Power failure. The executive calls the the device driver's power fail entry point under three conditions :

5.3.4.1. When power is restored and an I/O request is in progress or, optionally, on all occasions that power is restored.

5.3.4.2. When the system is bootstrapped. A power fail recovery is simulated whenever the system is first started, allowing the driver to carry out any required initialization.

5.3.4.3. When a loadable device driver is loaded. This is once more to allow initialization of the driver, and is optional.

5.3.5. Device interrupt. An I/O device may optionally have an interrupt (or two interrupts, for a full duplex device such as a serial line) associated with it. The device interrupt is entered when the I/O device requires service from the driver. This typically occurs when an previously initiated I/O operation ends.

5.4. Data structures

Two sets of data structures are associated with the device driver. These are firstly static data tables used to provide information on the device to the executive, and secondly dynamic

structures, created and deleted by the executive, containing information specific to a particular I/O request.

5.4.1. Static data structures. These structures are created, in code, at the time that the driver is assembled. Three structures are used, all of which are inter-related. These are the Device Control Block (DCB), the Unit Control Block (UCB) and the Status Control Block (SCB). RSX-11 recognizes devices (a particular type of peripheral, for example a specific disk drive), units (the individual peripherals) and controllers (a particular controller can control more than one unit). The device control block is used by the executive to obtain information on the characteristics of the device, the UCB to store information relating to each individual unit in the system, and the SCB to store status information for each device controller in the system. Strictly speaking, another structure is used, the interrupt vector, but this will be discussed when interrupt handling is discussed. Space does not allow a discussion of all the fields in each structure, so only the most important will be discussed here.

5.4.1.1. The DCB. At least one (and normally only one) DCB exists for each type of device in the system. Almost all of the data in the DCB is established by the assembly source code, and never modified. All the DCBs in the system form a linked list, and each DCB also contains a link to the driver code, as well as to the UCBs associated with the device. The driver itself does not normally use any of the information in the DCB, which is provided mainly to inform the executive of the characteristics of the device associated with it. The most important fields in the DCB are the following :

5.4.1.1.1. D.LNK. This is the link to the next DCB.

5.4.1.1.2. D.UCB. Link to the first UCB associated with the device.

5.4.1.1.3. D.DSP. Address of the driver dispatch table. This is a table containing the addresses of the driver entry points discussed above, and is contained in the driver code.

5.4.1.1.4. D.NAM. This is the two character name of the device eg. MT. All entry point names in the device driver code begin with these two characters.

5.4.1.1.5. Function masks. I/O functions under RSX-11 are encoded as an 8 bit code, so allowing 256 possible functions. For general I/O operations, of these only one, cancel I/O, is mandatory and only two, attach and detach unit, have fixed interpretations if used. For operation as a file device in conjunction with RSX-11, however, the first 32 function codes (0-31) have fixed meanings. In order to aid in the processing of these 32 functions, four bit by bit masks exist, each bit in the four masks corresponding to a particular function code. Each mask represents a particular type of function, and so informs the executive on the appropriate action to take for that function. Before a function request is passed to the device driver, the executive examines these four masks to determine what action should be taken. This process will be further described in

section 5.8.3. A bit representing a function is set in a particular mask if the function is of the mask type. Each mask contains 32 bits. The following masks, and hence function types exist.

1. Legal function mask. If the function bit in this mask is not set then the function is rejected.

2. No-op function mask. If the bit is set in this mask then no action on the part on the driver is required to execute the function.

3. Control function mask. This is a "normal" function. The I/O packet is created by simply copying the DPB information. Any function not represented in the function masks (not in the 32 most common functions) is assumed to be of this type.

4. ACP function mask. Functions with the ACP bit set are passed to an ACP for processing. If the legal function bit is set for a function, but neither the no-op, control or ACP bit is set then the function is a transfer function. Transfer functions are function which require a data transfer, and hence have a buffer address in the I/O packet.

5.4.1.2. The UCB. One UCB exists for each device unit in the system. Information in the UCB relates to each individual unit, and although established initially in

assembly source code is often modified by both the executive and driver. The UCB also contains a pointer to the SCB associated with it. The most important fields in the UCB are the following :

5.4.1.2.1. U.STS, U.ST2. These fields store the unit status.

5.4.1.2.2. U.CTL. The unit control flags. This stores various status bits, including the UC.NPR bit. If this bit is set then any buffer addresses supplied to the driver are formatted for DMA transfers, else the buffer addresses are formatted for program transfers.

5.4.1.2.3. U.UNIT. This stores the unit number to which the UCB refers.

5.4.1.2.4. U.CW1, U.CW2, U.CW3, U.CW4. These are control words. Their meaning is device dependant.

5.4.1.2.5. U.SCB. Pointer to the SCB for this unit.

5.4.1.2.6. U.BUF. Address of the data buffer for the operation (if any).

5.4.1.2.7. U.CNT. Size of the data buffer in bytes.

5.4.1.3. The SCB. One SCB exists for each device controller in the system. The SCB relates essentially

to the number of operations which can be performed simultaneously. If there is only one controller for a particular device in the system, only one operation can be performed at a time regardless of the number of units in the system. If on the other hand there are two controllers in the system, then two units can be operated on simultaneously, requiring two SCBs. Most information in the SCB is dynamic, although the structure itself is static. The most important fields in the SCB are the following :

5.4.1.3.1. S.LHD. The listhead of the I/O queue.

5.4.1.3.2. S.CTM, S.ITM. These are the initial and actual time-out counts. The initial time-out count (in seconds) is kept in S.ITM, and is copied to S.CTM when an I/O operation is begun in hardware. A time-out occurs when this reaches zero.

5.4.1.3.3. S.CON. Controller index.

5.4.1.3.4. S.STS. Controller status.

5.4.1.3.5. S.CSR. This is the base address of the controller hardware. The use of this field gives the device driver independence from the actual address of the hardware that it controls.

In the particular case being considered here, a single DCB is used to describe the characteristics of the tape units, and two UCBs are used as two tape transports are to be attached to the PDP-11. As there is only a single controller, and only a single serial link to it, a single SCB must be used.

5.4.2. Dynamic data structures. These data structures are created by the executive, and their operation is normally transparent to the driver. It is, however, necessary to have an understanding of them in order to fully understand the role of the device driver.

5.4.2.1. The I/O packet. The I/O packet contains information from two sources :

5.4.2.1.1. The QIO. Information on the requested I/O operation such as the function code and data address are copied from the QIO DPB (Directive Parameter Block) to the I/O packet.

5.4.2.1.2. The requesting task. Information on the requesting task such as its priority and the address of the task header are also contained in the I/O packet.

The I/O driver does not normally access the I/O packet directly, but uses executive service subroutines to extract information from it.

5.4.2.2. The I/O queue. Once the I/O packet described above has been generated it is inserted into a device specific I/O queue. The I/O queue is priority ordered, and one such queue exists for each device controller in the system. The SCB associated with the device stores the queue's listhead. As in the case of the I/O packet, the device driver does not normally manipulate the queue directly, but makes use of the executive service routines.

5.4.2.3. The fork list. The fork list is a mechanism by which RSX-11 synchronizes the access of interrupt driven processes to shared data bases. This is will be discussed further in the following section, which discusses interrupt handling under RSX-11 in general.

5.5. Resident and loadable device drivers

Device drivers may be either resident or loadable. The differences between these two types of driver will now be discussed.

5.5.1. A resident device driver is a permanent part of the executive, and can only be changed at system generation time. This means that if it is required to change the driver source code or the data tables, the entire executive must be rebuilt. As a result of this user written device drivers are almost always loadable.

5.5.2. A loadable device driver can be inserted into or removed from the system at any time. The use of loadable drivers has the following advantages :

5.5.2.1. Debugging is much easier as it is not necessary to rebuild the entire system (this takes, at best, several hours).

5.5.2.2. The load task, which inserts drivers into the system, does a variety of checks on the integrity of the driver and its data tables before the driver is loaded. This helps in detecting errors before they cause problems in the system.

The primary difference between resident and loadable device drivers is in interrupt handling. This will be discussed in the next section. It should be noted firstly that it is possible to have resident data tables and loadable driver code, and secondly that unloading a driver does not unload a data base, even if the data base is loadable. To remove the data base the system must be rebooted.

5.6. Interrupt handling under RSX-11M

As discussed earlier, each device controller can have associated with it an interrupt or, in case of a full duplex device, two interrupts, one for receive and one for transmit.

5.6.1. The interrupt vector. When an interrupt occurs in hardware the LSI-11 processor will acknowledge the interrupt, at which point the interrupting device will transfer the interrupt number to the processor. This interrupt number forms an index to a two word interrupt vector in memory. The address of the interrupt vector is calculated by multiplying the interrupt number by four. The first word of the interrupt vector is the address of the service routine, while the second word is used to replace the program status word in the LSI-11 processor. The status word contains two important pieces of information :

5.6.1.1. The priority of the process. The LSI-11 CPU can operate at one of eight priorities, 0 to 7. This priority is encoded in three bits of the status word, and when the status word is replaced a new priority is set. This priority is always set to 7 in the interrupt vector. As the CPU cannot be interrupted by a device

with lesser or equal priority a priority of 7 correspond to all interrupts being disabled. This allows the CPU to handle the interrupt without being interrupted itself.

5.6.1.2. The condition code bits of the status word (the lower four bits) are normally used to encode the number of the controller to which the interrupt refers. This immediately informs the device driver which hardware controller it is servicing, and so also which SCB it should obtain information from on the nature and status of the I/O operation in progress.

5.6.2. Establishing the interrupt vector. The interrupt vector is established in different ways for resident and loadable device drivers.

5.6.2.1. Resident drivers. For resident device drivers the interrupt vector is established at bootstrap time and cannot be altered later. The vector is coded in the assembly source code, and is inserted into memory at the the same time as the rest of the executive.

5.6.2.2. Loadable drivers. Loadable device drivers only have their interrupt vectors set at the time that the driver is loaded. The interrupt vector is calculated by the RSX-11M load task from information in the data base and in the driver task file. When a driver is unloaded the interrupt vector is set to the system nonsense interrupt. Note that a device driver can in fact only be loaded if the interrupt vector which it must use is set to this value. It should also be noted that because of the memory mapping of the driver the interrupt must be redirected via the executive for a loadable device

driver, and in fact this requirement necessitates the use of an executive service routine to handle interrupts, as will be discussed below.

5.6.3. Programming protocol. When a device interrupts, the device driver is entered at the interrupt entry point. At this stage the CPU priority is seven, and no other process can obtain CPU time. As this is a multi-user, multi-tasking system it is of critical importance to minimize the time spent in which other interrupts cannot be acknowledged. The RSX-11M operating system in fact provides three distinct levels of interrupt sensitivity.

5.6.3.1. Priority 7. This is the state described above. By convention processing at this level is limited to a maximum of 100 uS. If processing can be limited to this time, then the driver can simply process the interrupt, and return with minimum overhead. Normally however this is not possible and it is necessary to drop to a lower priority level.

5.6.3.2. The priority of the interrupting source. This level disallows interrupts from the same device as generated the interrupt, or any of lower or equal priority, but allows interrupts from a source of higher priority. Driver almost always make use of executive service routines to drop to this level, as this both simplifies coding and is in fact required by loadable device drivers. The service routine also routinely saves two of the CPU registers (R4 and R5). Processing time at this level should not exceed 500 uS.

5.6.3.3. Fork level. Interrupt handlers often require more time than is allowed by the level described above,

or they require access to common data areas which might be corrupted by unrestricted access. In general there are two possible systems for controlling access to data areas by interrupt driven software.

5.6.3.3.1. Interrupt lockout. This system simply involves disabling all interrupts until the present interrupt has completed processing. This system has the advantage of being simple, and is the most efficient of the two systems, but is incapable of meeting the response time requirement of the system.

5.6.3.3.2. Priority queuing. This system involves operating the interrupt driven software in the uninterruptable mode for only as long as is absolutely required by the nature of the device, and then queuing the process to allow it to complete processing at a later time. RSX-11M provides a mechanism called a fork list implement this system.

A process becomes a fork process by calling the executive routine \$FORK. When this occurs the executive stores a "snapshot" of the process, and queues the process in a first-in-first-out queue. Once the process has worked its way to the head of the queue the process is restored, and has unrestricted access to all common data areas. The process exists in a state between that of an interrupt process and that of a normal task. A fork process is fully interruptable, but the process is guaranteed sole access to the common data areas as all other process which might wish to access such areas must also make use of the fork list to obtain access.

This technique allows the interrupt process to spend a minimum of time in the uninterruptable state, but still get uninterrupted access to the system data areas. Note that no normal tasks will be run until the fork queue is empty. It should also be noted that this technique assumes that all code in the system is well behaved, and does not make accesses to system data tables unless \$FORK has been called.

5.7. Executive services available to device drivers

In the sections above mention was made on several occasions of executive services available to the device driver. The most important of these services will now be discussed briefly.

5.7.1. Get packet (\$GTPKT). Once an I/O packet has been queued, the executive calls the device driver. The driver then calls \$GTPKT to obtain work. If work is available the executive dequeues the highest priority packet and sets the controller to busy. Information on the I/O request is also copied to the UCB of the device for which the I/O request is intended. If no work is available \$GTPKT returns an indication of this, and the driver returns control to the executive. Note that no distinction is made between no work because the controller is busy, and no work because the I/O queue is empty.

5.7.2. Get byte (\$GTBYT) and put byte (\$PTBYT). These two routines respectively get and put a byte of data to and from the specified data buffer. The address of the next byte is stored in the UCB, and these routines update this for the next operation.

5.7.3. Interrupt save (\$INTSV). This routine is called by the driver in order to drop it's priority to that of the interrupting device, and to save CPU registers 4 and 5. This routine is normally called via the INTSV\$ system macro, which automatically adjusts the call parameters to suit either a resident or loadable device driver. The INTSV\$ macro also sets the mapping of the LSI-11 memory management unit.

5.7.4. Fork (\$FORK). This inserts the calling process into the fork queue, as discussed above in the section on interrupts. The driver must have called \$INTSV before calling \$FORK.

5.7.5. I/O done (\$IODON). At the completion of an I/O request the \$IODON routine is called to inform the executive that the I/O request is complete. The driver returns two words of status information, describing the results of the I/O operation, which are returned to the task which issued the I/O request. The routine also sets the SCB status to idle. When \$IODON completes, the driver executes a call to \$GTPKT to see if there is more work in the queue, and only if the queue is empty does it return control to the executive.

5.8. Flow of an I/O request

In order to describe how the various parts of the RSX-11M I/O system fit together, the flow of a typical I/O request will now be described. For the purposes of this discussion it will be assumed that this is the only current I/O request, and that no errors are encountered.

5.8.1. The task issues an I/O request. The executive checks to see that the I/O request has been directed to a valid unit, and that no redirection is required. If redirection is required the executive traces the redirection path until the target device's UCB is located. Various other checks on the validity of the I/O request are then done, and the status return area is cleared.

5.8.2. The I/O packet is created. The executive obtains an area of dynamic storage, and creates the I/O packet from the I/O request's DPB (Directive Parameter Block).

5.8.3. The executive validates the function. The requested function is one of four types, as discussed in the section describing the DCB. The DCB's function masks are used to decide the function type, which is then processed further as described below.

5.8.3.1. Control functions. These are simply queued to the driver.

5.8.3.2. No-op. A function which is set as a no-op does not require driver processing, and is hence simply processed to completion by the executive.

5.8.3.3. ACP. ACP functions are queued to the appropriate ACP, which then reissues the request as a series of functions which can be performed by the driver.

5.8.3.4. Transfer functions. Transfer functions are checked for valid data buffer addressing, and then queued to the driver.

5.8.4. Driver processing. Once the above steps have been accomplished the driver is called, which then calls \$GTPKT to obtain work. Once the driver obtains this work it initiates the required operation in hardware, and returns to the caller to wait for an interrupt. Operations not requiring interrupts continue directly from this point to the I/O done section (see 5.8.6.).

5.8.5. Interrupt processing. When the I/O process interrupts, the interrupt causes a direct entry into the interrupt code of the driver. The driver will then go through the various levels of interrupt processing as described above, and either continue to the next section, or issue a further I/O operation in hardware and continue to wait for this to complete.

5.8.6. I/O done processing. When the I/O operation is complete the driver calls \$IODON as described above. The executive returns the results of the I/O operation to the requesting task, and the driver is then ready for the next I/O request.

CHAPTER 6

THE DEVICE DRIVER

6.1. Introduction

This chapter discusses in detail the device driver written for the tape transport controller. The previous chapter provided a basic description of RSX-11M, and of the general requirements for any device driver. This chapter will concentrate on the requirements of the particular driver required for the tape transports, and will also lead in to the discussion of the software imbedded in the actual tape transport controller.

6.2. Requirements for the driver

Several requirements for the device driver have been mentioned in previous chapters. These are listed below, as are several other features which would be desirable :

6.2.1. The driver must, of course, meet the requirements for device drivers under RSX-11M as described in the previous chapter.

6.2.2. The device driver must, as far as possible, emulate a known tape transport/controller combination. The controller chosen for this purpose was the TM-11 controller, manufactured by DEC. This emulation can be summarized as follows :

6.2.2.1. The device driver must execute the same set of instructions as the original controller.

6.2.2.2. The device driver must provide the same status returns as the TM-11 controller. It is very important that the driver should return both the same status words and should affect the tape units characteristics bits in the same way, under both normal and error conditions. In practice we can expect that this goal will not be totally achievable as the TM-11 and the controller to be designed are physically very different.

6.2.2.3. The device driver must appear identical to MT-drive (the device driver used in conjunction with the TM-11 controller) as far as the operating system is concerned. As discussed in the previous chapter, the executive obtains information on the characteristics of the device from the drivers data tables. These tables should obviously be the same as the TM-11's tables, or at least as close as is possible, given the differences between the devices.

6.2.3. The device driver should be as compact as possible. There are several reason for minimizing the drivers code size :

6.2.3.1. Limited space is available for loadable device drivers. All loadable device drivers under RSX-11M are mapped onto kernel APR5. As each memory mapping register can access only 8K bytes of memory all loadable device drivers in the system must have a total size of less then 8 K bytes, or must modify the system memory map when they execute. As altering the system's memory map is both difficult and time consuming it is desirable that the driver should take up a minimum of storage space.

6.2.3.2. The device driver should use as little of the PDP-11/23's CPU time as possible. Much of the drivers code is executed with interrupts locked out, making it important to minimize code size, and execution time.

6.2.3.3. It is also desirable to have a minimum of code to debug. As driver code is always privileged, and executes as part of the executive, any error will almost inevitably cause a system crash. Another factor to be considered is that because the driver executes as part of the executive one runs the risk of introducing subtle bugs into the system which can cause crashes apparently unrelated to the driver.

6.2.4. The driver should be coded in such a way as to maintain DEC's coding standards. There are two reasons for this. Firstly although we have discussed only RSX-11M, several other operating systems and versions of operating systems exist to run on several different versions of the

basic PDP-11 processor. Maintaining DEC's coding standard eases the task of porting the device driver to another system. A typical example of one of these considerations is that of memory management. Conventionally device drivers do not manipulate memory mapping registers, but leave this to the system utilities described in last chapter. This ensures that the device driver will not need to be rewritten to run on a system which does not provide memory mapping. A second reason for maintaining DEC's coding standard is to allow future programmers to easily compare this device driver to standard device drivers, and so ease the task of possible updates at a later date. For this reason the same general layout and form of commenting was used for the device driver as is used in standard system drivers.

6.2.5. The driver must provide the protocol required to ensure that the data link to the tape controller is reliable. The data link between the host computer and the tape controller is a serial link. It is the responsibility of the device driver to control this link. This aspect of the device driver will be discussed in detail in the following section.

6.3. The serial link

6.3.1. Objectives. The serial link must transfer blocks of data between the PDP-11/23 and the tape controller. A typical sequence would involve the PDP sending a command block (for example read block) and the tape controller responding with status information and any data. A protocol must be devised which will allow this block level transfer to take place. Ideally, this would simply involve the host transmitting it's command block, and then receiving the controller's reply block. In practice several

problems relating to the characteristics of both the physical line and the host computer must be overcome.

6.3.2. Characteristics of the serial link. The serial data link between the host system (the PDP-11/23) and the tape controller is an RS-232 link. The full RS-232 specification makes provision for several lines, including several for flow control. In practice most of these lines are not used. The ground, received data and transmitted data lines are the only lines implemented by the serial interface in use on the PDP-11, as well as being the only lines wired from room to room in the department's data distribution system. The device driver must fulfill two requirements as regard the serial lines :

6.3.2.1. Flow control. Any real computer system can only accept a certain amount of data in a particular amount of time. It is thus necessary to provide some form of control over the flow of data from one device to another. In some RS-232 links this is done by the hardware lines mentioned above, but for most systems (including the PDP-11) this is done by software. A typical system which is widely used is the so called XON/XOFF protocol. This works essentially by having the receiving device transmit an XOFF character to stop data transfer, and an XON character to resume data transfer.

6.3.2.2. Error control. Under ideal conditions data could simply be put onto a line, and would emerge at the other end in exactly the same form. For real lines this is not so. For the PDP-11 in particular two main problems exist :

6.3.2.2.1. Line noise. Any real data line is subject to errors due to electrical noise, requiring some form of error detection/correction. RS-232 lines are unbalanced, and so especially sensitive to ground

noise. Line noise normally will result in one or more bits in a character being corrupted. This is normally detected by a parity bit in each character, and most USART chips can be programmed to include a parity bit automatically. Parity bits can however only detect a single bit error per character.

6.3.2.2.2. Lost characters. In the case of the PDP-11 (and most multi-user systems) an additional problem arises. All serial interfaces interrupt at the same priority, and hence a new interrupt will not be recognized until the device driver becomes fully interruptable. As discussed in the previous chapter a device driver is allowed to operate at the priority of the interrupting device for 500 μ S. A serial line operating at 9600 baud delivers characters at approximately 1 mS intervals. This means that at most TWO serial devices can operate at full speed under RSX-11M assuming that the device driver uses its full allocation of time. As the PDP-11 currently has 24 serial lines attached it is obvious that if several lines are operating at full speed (for example running file transfer programs) characters will be lost. Equally obviously error detection by character parity will not help, as entire characters will be lost.

By far the most commonly used solution both to flow controlling the link and to error control is to acknowledge either each character, or each group of characters. Acknowledging each character has the advantage of simplicity and a very high level of reliability, but is very inefficient. This technique is in fact used by DEC for some PDP-11 peripherals (such as DD-drive, a dual tape cassette system) as it is very effective for dealing with the PDP's

lost character problem as described above. Acknowledging groups of characters (typically 64 to 256 per group) is far more efficient, but requires more complex software. In this case it was decided to use group acknowledgement, in order to maximize link throughput. These groups are normally known as frames and their related protocols are known as frame protocols. As digital data transmission has grown in importance so much research has gone into various forms of frame protocol. It is not the purpose of this thesis to become embroiled in an analysis of all the available techniques, and so only the technique to be used will be discussed here.

6.3.3. Line protocol. Most modern digital data communication systems distinguish between several levels or layers in the system. The basis of this approach is to have the same layers in both the receiver and transmitter. Communication can then be modeled at each level as a link to the level in question's opposite number in the receiver. Layers below the level in question are viewed as a virtual communications channel, the characteristics of this channel depending on the layer under discussion. This process culminates in the lowest level, known as the physical layer, in which this link is in fact a physical link. This technique has the significant advantage that layer can be changed, as long as the layer to layer interface is constant. This approach is also often used to aid in understanding a communication system, and indeed several standard representations (such as the ISO model) [14] have been devised.

6.3.3.1 Layers. In this case it is useful to recognize four layers, although our definitions will bear only superficial resemblance to most generally used models :

6.3.3.1.1. The physical layer. This layer is the actual physical interface between the host and the controller. At present this is the RS-232 link as described above.

6.3.3.1.2. The link layer. This layer is involved with the transmission of frames. This is done by means of the physical layer. The responsibilities of the link layer include error detection and flow control. Complete frames are passed to the next higher layer, together with error information. Note that although the frame may contain errors this will be signaled to the block layer for appropriate action.

6.3.3.1.3. The block layer. This layer makes use of the link layer in order to transmit and receive complete blocks of data. Blocks consist of an arbitrary number (limited only physical constraints of available memory) of bytes of data. This layer passes complete error free blocks to the application layer, and is roughly equivalent to the transport layer in the ISO model [15].

6.3.3.1.4. The application layer. This is the layer which actually uses the data. In the case of the host system this is the operating system, and in the case of the controller this is the module which translates host commands into tape motion.

The split as described above is not exactly in accordance with the layers of the ISO model [14,15], but was designed for maximum efficiency in this particular application. Ideally each layer (except the physical layer) should be implemented as a separate layer of software. In the controller this is done, but in the device driver this would

be futile as the device driver is already the lowest level in the operating system, itself a layered system.

6.3.3.2. The link layer. The physical layer has already been discussed in section 6.3.3.1., and it remains now only to describe the link layer and the block layer. The link layer has the following features :

6.3.3.2.1. Each block of data (in this case a command to, or reply from the tape controller) is split into a number of frames. In general these frames may be either of fixed size, or of variable size. Fixed frames are padded with dummy characters where necessary, and the frame normally contains a field specifying how much of it is "real" data. Variable length frames use some predefined sequence of characters to denote the end of the frame. In this case it was decided to use variable length frames as the majority of the commands and responses required are short (typically six bytes) and the use of fixed length frames would result in the transmission of a large amount of unused information. Data transparency is achieved by character stuffing. Each frame thus consists of control characters and from zero to some maximum number of characters. An optimal value for this number may be calculated, based on factors such as the error rate of the channel and the time taken to recognize that an error has occurred (indeed this may be done adaptively) but in practice some reasonable number is normally selected. In this case a maximum frame size of 64 characters was chosen firstly in order to minimize the size of the local frame buffers in the device driver, and secondly to minimize the time taken to recover from a line error.

6.3.3.2.2. Operation of the link is identical at the frame level regardless of whether the block to be transmitted originates from the host or the tape controller. The sequence of events for a frame transfer is as follows. A data frame is transmitted by either the host or the tape controller. When this is received the receiving device transmits an acknowledge frame. Acknowledge frames are special purpose frames which carry no data but are used only to reply to data frames. The acknowledge frame may be either a positive or negative acknowledge depending on whether the data frame contained any errors. If the transmitting device receives a negative acknowledge frame the data frame will be retransmitted. The transmitting device also contains a timer, which is used to retransmit a frame if no acknowledgement is received within some specified time. If a positive acknowledge is received then the next frame is sent, unless the previous frame was the last in the block. This is known as an Automatic Request Repeat Protocol.

6.3.3.2.3. Each data frame also contains a frame number. This is to prevent data from being duplicated in the received block in the event of positive acknowledge frame being lost or corrupted, resulting in a retransmission of a frame which was correctly received. This frame number is counted modulo two as only one frame can be outstanding at any one time. Note that each acknowledge frame contains the number of the frame which it is acknowledging.

6.3.3.2.4. Errors are detected by a combination of three techniques. These are as follows:

1. Line error detection. This involves the detection of errors such as overruns and framing errors. These errors are detected by the hardware used to send the data stream.
2. Invalid character sequences. As discussed earlier the start and end of frames are signaled by particular character sequences. Any invalid sequence results in the frame being discarded.
3. CRC codes. Each frame has, as the last two characters, a 16 bit CRC. The polynomial chosen is CCITT-16, which detects all burst errors of less than 16 bits, and more than 99.99% of all other errors [15].

Further error detection capability could be added by transmitting parity information with each character. This was however not implemented as not only would parity be of questionable value in light of the error detecting capability of the CRC, but it would require modification of internal jumpers in the PDP-11 system, and so make the serial channel in question incompatible with conventional terminals.

A complete description of the format of all frames may be found in appendix D.

6.3.4. Block format. The link level protocol described above allows the transfer of an arbitrary block of data to and from the tape controller. Note that a block originating from the controller will always be in response to a block from the host. Note also that the format of command blocks (to the controller)

and response blocks (from the controller) must be different, but that this is, of course, transparent to the frame protocol. Obviously it is necessary to decide on the format for command and reply blocks. As discussed earlier, it was decided to do as much as possible of the required processing in the controller in order to unload the host system to as great an extent as possible. To this end it was decided simply to transfer the contents of the PDP-11 I/O packet to the tape controller with as little manipulation as possible. Similarly the reply block contains status returns in a format as near as possible to that required by the PDP-11's I/O system. This results in block formats as follows :

6.3.4.1. Command block. The command block consists of the following fields :

6.3.4.1.1. Opcode. This is an 8 bit byte which contains a 6 bit command code (such as read block, write block etc.) as well as two bits which indicate whether retries are suppressed, and whether or not the mounted tape is ANSI formatted.

6.3.4.1.2. Unit number. This is a byte which indicates the logical unit number of the tape transport to which the command refers.

6.3.4.1.3. Tape status in. This word contains the status byte of the tape unit in question. The contents of this word are as described in appendix D.

6.3.4.1.4. Count in. This word can contain three different pieces of information, depending on the command. These are as follows :

1. For read and write operations this field contains the byte count for the operation. Note that for read operations this is the expected count value, and if the actual value is greater than this an error must be reported.

2. For space operations this field contains the number of files or blocks to be spaced in twos complement form.

3. For status set operations this field contains the new status value. Note that only certain bits may be altered by the operating system. The controller must for example prevent the operating system from attempting to operate the tape drives as seven channel units.

6.3.4.1.5. The next and last field contains data for write operations. This may consist of up to 8192 bytes of data. This field is not used for any other command.

6.3.4.2. Reply block. The reply block consists of the following fields :

6.4.3.2.1. Tape status out. This word contains the status byte of the tape unit in question. The contents of this word are as described in appendix D.

6.4.3.2.2. Count out. This word can contain two different pieces of information, depending on the command. These are as follows :

1. For read and write operations this field contains the byte count for the operation.

2. For space operations this field contains the number of files or blocks to be spaced in twos complement form.

6.4.3.2.3. Return code. This field contains the return code for the operation which is passed to the operating system. The various possible returns are described in appendix D.

6.4.3.2.4. The next and last field contains data for read operations. This may consist of up to 8192 bytes of data. This field is not used for any other command.

6.4. The device driver code

6.4.1. The device driver code consists of two distinct sections, the first being the executable code (the driver itself) and secondly the driver's associated data tables. These two sections will now be described. The description below should be read in conjunction with the listings in appendix E and the discussion of the device driver's role in the previous chapter.

6.4.1.1. The executable code. The executable code may best be described in terms of the various entry points as described in the previous chapter.

The software is simple in concept. The sequence of events is as follows, assuming that only one I/O request is active,

and that no errors occur. An I/O request is generated and the I/O driver called. After initialization, the first frame is transmitted. Once this is acknowledged the next frame is sent. This process continues until the last frame in the command block has been transmitted and acknowledged. The device driver then waits for the first frame of the reply block, and acknowledges this and any subsequent frames. Once the last frame in the block has been received the driver passes the contents of the reply block to the executive, and waits for the next I/O request.

The most important part of the device driver is that which is responsible for the frame protocol. It has been shown [15] that in general the most reliable and testable technique with which to implement a communications protocol is via a finite state machine. This has the advantage that as the machine can only have a finite number of states, testing is limited to ensuring that the machine responds correctly to all possible input conditions in each state. A further advantage of this approach is that only the code directly concerned with any particular state is executed in that state. This normally results in code which is close to optimal in terms of execution speed and size, without the programmer having to pay undue attention to these aspects in the program design. This approach has been taken for both frame transmission and reception.

A detailed description of the code will now be presented, in terms of each entry point :

6.4.1.1.1. I/O initiator. This entry point, labeled MAINI in the listing, is the point at which the executive calls the the device driver. Processing proceeds as follows :

1. The device driver executes a call to \$GTPKT. \$GTPKT first checks to see if the requested device is busy, and that there is work for it. It is necessary to check for work as the driver executes a jump to this entry point on completing an I/O request to check for further work.

2. If there is work then the driver proceeds to check the request for validity. The labeled tape and retry suppress bits, which are transmitted to the controller as part of the opcode field, are first set up, and then the opcode is checked against a table of valid opcodes. Assuming the request to be valid for magtapes, the driver then proceeds to check that the operation is not invalid due to a previous power failure. If the operation is a rewind then the power fail bit is reset. The opcode is then added to the labeled tape and retry suppress bits set up earlier, ready for transmission.

3. The device driver then proceeds to set up to transmit the block to the controller. The opcode, tape status and count value are copied to the frame buffer, and the code falls through to the frame setup section.

4. The frame set up routine (at label NEXTFRM) proceeds to set up the frame by filling the frame buffer with data (if any), and setting the frame start and stop characters. Note that NEXTFRM is entered only once the previous frame has been successfully transmitted.

5. The frame send routine (at label SENDFRAME) transmits the frame in the frame buffer. Note that this routine is used for both data and acknowledge frames. Executing this routine without setting up a new frame results in the retransmission of the previous frame. If the frame is a data frame then the frame retry counter is checked to see that the maximum retry limit for the frame in question has not been reached. If this limit has been reached then the driver will return an error code to the executive. Note that this routine also sets up to receive a reply frame, as well as setting up the timeout count value described in section 5.4.1.3. If a timeout occurs, then control is passed to the timeout entry point described in section 6.4.1.1.7. Once this set up procedure has been completed the routine checks to see if the transmitter is busy. If so the transmit interrupt is enabled and the driver returns control to the executive, and waits for an interrupt indicating that the transmit data port is empty. If the transmitter is not busy then the first character of the new frame is placed in the transmit data port, following which interrupts are enabled and a return executed. In either case further processing is via the interrupt system. This will be described in the next section.

6.4.1.1.2. The output interrupt. This interrupt occurs on each occasion that the transmit data port becomes empty, assuming that the the transmit interrupt is enabled. When this interrupt occurs processing proceeds as follows :

1. The priority is immediately dropped to that of the interrupting device. Once this is done, the base address of the transmit data port is obtained from the SCB. This is done to allow the device driver code to be independent of the actual hardware address, as well as to allow the driver to operate several identical device controllers, each of which will reside at a different hardware address. Note that in this case no provision has been made for operation with more than one controller, although the device driver code could easily be modified to provide this capability. Note that this means that in order to change the serial port used by the tape controller only the data tables, and not the driver code must be modified.

2. The driver then executes an indirect jump via the state table, indexed on the transmitter state. This transmitter state value is held in storage local to the device driver, and is modified on each pass through the device driver. This indexed jump is exactly analogous to a high level language's case statement as follows :

```
case transmit_state of .....
```

Execution then proceeds to the appropriate section of code for the current state. These states are discussed in appendix D.

The code specific to the state in question stores the character to be transmitted on the stack, and a jump is executed to the common exit code, which

transmits the character in question, as well as calculating the CRC using a logic operation technique similar to that described by Perez [23]. This technique is used as it provides an optimal combination of compact code and speed of operation.

3. Once the entire frame has been transmitted two possible courses of action are available to the driver :

3.1. If a response is required from the controller the driver will enable the receive interrupt and wait for a response frame.

3.2. If the previous frame was the acknowledge frame for the last frame in a reply block then the driver will proceed to its final processing routine. Only if this is the case is a fork executed. This will be described in a later section.

6.4.1.1.3. Input interrupt. This interrupt occurs on each occasion that the receive data port becomes empty, assuming that the the receive interrupt is enabled. When this interrupt occurs processing proceeds as follows :

1. The priority is immediately dropped to that of the interrupting device. The base address of the receive data port is obtained from the SCB, and the received character is read in. This occurs exactly as was the case for the transmit

interrupt. The CRC is then calculated using the same logic operation technique used for the transmit interrupt.

2. The driver then executes an indirect jump via the state table, indexed on the receiver state. Processing then proceeds in a similar fashion to that for the transmit interrupt. The various states are discussed in appendix D.

3. Once a complete frame has been received it is examined to determine the appropriate action as follows :

3.1. If the received frame is a negative acknowledge then a branch to SENDFRAME is executed in order to repeat the previous frame.

3.2. If the received frame is a positive acknowledge then a branch to NEXTFRM is executed in order to transmit the next data frame. If no further data is to be transmitted then the the driver will wait for the first frame of the reply block.

3.3. If the received frame is a data frame then if there is an error in the frame a negative acknowledge frame is set up and a branch to SENDFRAME executed to send this. If there was no error then a positive acknowledge frame is set up, the data in the frame copied to the appropriate place (the

first three words of the first frame to status storage, all other data to the block buffer) and a branch to SENDFRAME executed.

6.4.1.1.4. Final processing. Once the acknowledge frame for the last data frame in the reply block has been transmitted, then the status information previously stored is loaded into the appropriate registers, and a call to the executive routine \$IODON is made. This informs the executive that processing of the current I/O request is complete, and returns the results of the I/O operation. A branch to MAINI is then executed to check for further work, as described above.

6.4.1.1.5. Driver cancel entry point. This function (called at label MACAN) is basically ignored by magnetic tape drives. However if a timeout occurs after a call to the cancel entry point then the I/O operation will be aborted. This allows the user to abort an I/O operation more quickly than would be the case if it was necessary to wait for a complete set of retries to be attempted. The cancel entry point simply sets a status bit which may be examined by the timeout code.

6.4.1.1.6. Power fail entry point. On entry to this routine the power fail status bit is set, but only if this feature is supported by the executive. The code at this entry point is only included if the global symbol P\$\$RFL is defined. This symbol resides in the system definition file, and indicates the executive configuration.

6.4.1.1.7. Timeout entry point. If the time out count set up as described in section 6.4.1.1.1 reaches zero, then the executive executes a call to this entry point. The code at this entry point checks the status bit referred to in section 6.4.1.1.5. to see if an abort was requested as described above. If an abort was requested, it aborts the operation, returning a status code indicating this to the executive. If an abort is not required then the previous frame is retried unless the maximum number of retries have been exceeded.

6.4.1.2. The data tables. As discussed in the previous chapter it is essential that the data tables for the device driver should be as similar as possible to those for the MTDRV, as the executive obtains most of its information on the characteristics of the device from these tables. In practice it has been possible to make MADRV's data tables identical to those of MTDRV, with one exception. This is in the setting of the UC.NPR bit in the UCB. This bit indicates to the executive in which of two formats the address of the data buffer should be presented to the driver. If this bit is set then the presentation is suitable for an NPR device (that is a No Processor Request device, DEC's way of referring to a device which transfers data by way of DMA), and if it is not set then the presentation is suitable for a device which will make use of the executive routines for data transfer. As MTDRV makes use of DMA this bit is set in its data tables. In the case of the driver under discussion here, program transfer techniques are used and so the bit must not be set.

6.4.2. Building the device driver. The device driver is assembled and built as described in reference 16. A complete listing of the commands used is supplied in appendix G.

6.5. Testing the device driver

6.5.1. This section describes only the initial testing which could be done without the use of the tape controller. Information on the testing of the entire system will be presented in a later chapter. The initial testing may be split into several steps :

6.5.1.1. Loading the device driver and it's associated data tables. As discussed earlier, the processor used to load the device driver executes a variety of checks on the validity of the the driver and it's data base.

6.5.1.2. The validity of the data tables were further checked by displaying the status of the driver by the MCR command DEV. This displays the status of the driver, and any devices associated with it, based on the information found in the data tables.

6.5.1.3. The MCR was used to generate I/O requests to the driver, and the output on the serial line monitored on a terminal. The basic operation of the transmitter routine was verified in this way. Since no response was generated, the operation of the timeout facility was also tested.

6.5.1.4. As the next step, reply frames were manually generated on a terminal. Both correct and incorrect frames were simulated, and the response of the driver verified.

6.5.1.5. Finally a program was written (in RTL2) which allowed the generation of I/O requests with arbitrary function codes, as well as to display the drivers status returns. This allowed the verification of the driver's

ability to reject illegal commands, as well as checking the status as returned by the driver for various I/O operations.

CHAPTER 7

THE TAPE CONTROLLER SOFTWARE

7.1. Introduction

7.1.1. This chapter discusses the software which is imbedded in the tape controller. Previous chapters have described the hardware which makes up the tape controller, as well as the software used by the host system to communicate with the controller. The description of the imbedded software has been left until last as the software can be viewed as an interface between the host software and the controller hardware. This view is especially apt as, for reasons described in the previous chapters, both the host software and the controller hardware were designed to be as simple as possible, and the maximum number of functions left to the controller software.

7.2. Requirements

The technical requirements for the imbedded software have in the main been defined by the previous chapters, which described the

hardware and the host software, and need not be repeated in detail here. The main points are however worth repeating, together with some new material.

7.2.1. Firstly, and obviously, the software must provide the emulation of the tape transport discussed previously.

7.2.2. The controller must also provide diagnostics functions. These should not only be included as a matter of good engineering practice, but will be indispensable should the controller ever be upgraded to operation on a LAN.

7.2.3. Facilities to allow the user to execute certain operations on the tape under local control are desirable. A major disadvantage of the serial data link used to connect the controller to the host computer is speed. This problem can be substantially reduced by allowing the user to access the tape controller in local mode. For example, tapes can be duplicated at far higher speed in local mode than if all data first had to be transferred to the host and then back to the controller.

7.2.4. The software written for the controller should follow good software engineering practices. Since the introduction of microprocessors the importance of software has increased dramatically both from the point of reliability and cost. As the number of products which incorporate microprocessors have increased, so has the realization among project managers that software has become both the most significant reliability problem, as well as the worst controlled cost factor. This has lead to a general desire to use the principles of engineering to improve programming practice. The next section will discuss this, among other issues.

7.3. Software structure

7.3.1. Use of a high level language. It has been shown conclusively [19] that programming effort, as well as the number of bugs in the completed code are directly related to the number of source lines in a program. Programming effort may in fact be modeled as follows :

$$MM = K_m(KDSI)^{K_e} \quad (7.1)$$

where

MM = Effort in man-months.

K_m = Multiplicative factor, the value of which depends on the type of software and the model in use. For advanced models this factor is the product of several factors K₁..K_n, where each K_n represents a different factor (or driver, for example programmer competence).

KDSI = Number of thousands of delivered source instructions in the final product.

K_e = Exponential factor. This is also dependant on the type of software and the model in use. Factors of greater than one indicate a diseconomy of scale, and factors of less than one indicate an economy of scale.

Several models exist of this general form (at least 11 to the author's knowledge), but the choice of variable names above is for a generalized version of COCOMO (Constructive Cost Model)[19], of which several specific versions exist, each with different coefficients. COCOMO is claimed (by its authors) to be the most accurate of its type, and is backed up by a considerable

amount of statistical data. It should be noted that certain restriction exist for this model, notably that is it not useful for code sizes of less than approximately 2000 DSI. We will only discuss the basic COCOMO model here. For this model three modes of software development are recognized.

7.3.1.1.1. Organic mode. The major characteristics of the organic mode are that the specifications of the project are relatively flexible, and that the environment is relatively stable. An example of this would be the reduction of experimental data on a mainframe. For this mode $K_m = 2.4$ and $K_e = 1.05$.

7.3.1.1.2. Embedded mode. The major characteristic of the embedded mode is that the specifications of the project are very tight, normally defined by hardware or other essentially unchangeable characteristics. An example of this would be an aircraft collision avoidance system. For this mode $K_m = 3.6$ and $K_e = 1.20$.

7.3.1.1.2. Semidetached mode. This mode is a combination of the first two modes, and is used where the project is a combination of organic and embedded mode characteristics. An example of this would be an operating system. For this mode $K_m = 3.0$ and $K_e = 1.12$. This project can probably best be modeled in semidetatched mode, although the embedded mode characteristics of the project predominate. Note that the more advanced versions of COCOMO do not make the rather artificial split into modes, making use of various drivers to provide a single model which is continuous.

An examination of equation 7.1 leads directly to the use of a high level language, in order to minimize the number of source instructions. At the time that this project was begun the choice

of languages was limited to one, Intel's Pascal-86. Fortunately this language is very well suited to use in this role, as it produces compact code, and makes special provision for producing software which is modular. In practice the use of high level languages for embedded software, and especially for systems in which the code is to reside in ROM, has not proved as successful as might be hoped. This stems largely from the difference between the use of these languages on a host system, and their use on a target system.

Program development is normally done on some form of a host system (in this case an Intellec development system) which runs editors, compilers, linkers etc. Once the program has been developed it is transferred to the target system (in this case the tape controller). Very often it is possible to execute the high level code on the host, either because the host contains the same CPU as the target system (as the Intellec does) or by making use of another similar compiler which generates code for the host. Unfortunately there are certain problems related to the use of high level languages in applications requiring imbedded software :

7.3.1.2.1. High level languages can often not be used to provide all required functions. For example Pascal-86 does not allow access to the address of a variable, which is required for programming DMA registers for the timing module. Once a programmer is forced to write even a small amount of code in assembler it is a great temptation to write all the code in assembler.

7.3.1.2.2. Bad compiler documentation. Most high level languages are used on a host system, and are hence written for an environment where the entire program is in RAM, and all IO is done via an operating system. This results in

compilers which are not documented for use in any environment other than the host, and testing is also often perfunctory for applications in environments other than the host. During the development of this project two versions of Pascal-86 were used, both of which had serious flaws in their documentation as regards use in a target system other than the host, and one of which (the later version!) contains a bug which renders code generated under a particular memory model useless for programming into ROM. The faults are described in detail in appendix H.

7.3.1.2.3. Many of a compiled language's advantages disappear when used on a target system. In the past there has seldom been any attempt to install the language's run-time system on a target system, so seriously reducing the high level language's usefulness. A language's run-time system is responsible for all environment dependant features of a language such I/O, error handling and memory management. The ease of use of a high level language may to a large extent be ascribed to three factors, each of which requires the use of the language's run-time system :

1. Run time error messages. When using a compiler on a host system the user may expect run time error messages (divide by zero etc.) which aid considerably in fault location, as these not only tell one what happened, but also often contain information as to where the error occurred (line number etc.). When a high level language is used on a target system however an error normally results in a system which simply ceases to respond.

2. Use of I/O library routines. When used on a host system, I/O library routines are used to allow data transfer. These library functions free the user from

the task of format conversion, as well as making it very simple to insert trace instructions to locate bugs. When a high level language is used on a target system not only must the user write format conversion routines, but the inclusion of trace instruction can be a major undertaking.

3. Standardization. When a high level language is used on a host system the user can normally write a program in a standard form, primarily by making use of the I/O library discussed above. This has two advantages :

3.1. Familiarity. The programmer will be familiar with the instructions used, and as discussed above will be able to use standard instructions to obtain trace information.

3.2. Modules may be tested on the host system. If only standard instructions are used in a module then the module may be tested on the host system, without the requirement to transfer the software to the target system. This is not only speeds up program development, but allows the user to make use of debugging tools on the host, which are not normally available on the target system.

7.3.1.3. There is little to be done about the problems of compiler suitability and documentation described above, but a technique does exist to retain the advantages of standardization and I/O library use discussed above. All of Intel's compiled languages make use of a common run-time system, which may be replaced by the user. This allows the user to effectively install the language on a target system, and hence have access to all the language features which

depend on the run-time system. The run time system for Intel languages consists of three sections, arranged in a loose hierarchy :

7.3.1.3.1. Run time library. This section, which is always present, contains the all support code which is required by the language under all circumstance, such as the procedures used to support various language features, for example set manipulation. This section is language specific, and provides the interface between the language and the other two sections, which are the same regardless of language.

7.3.1.3.2. Logical Record System (LRS). This section provides I/O, memory management and error handling facilities to the language. Intel provide two LRS modules for their languages. One is designed for use on systems which contain an operating system, and provides run-time support by calls to the Universal Development Interface (UDI) which will be described in the next section. The second Intel supplied LRS is designed for use on a bare machine, and provides no run-time support. This is known as the null version, and simply provides status returns indicating that run-time facilities are not available.

7.3.1.3.3. Universal Development Interface (UDI). This section of the run-time system provides similar services to that of the LRS, with the addition of functions which are specific to an environment which contains an operating system, for example obtaining parameters contained in the command line which invoked the program. Note that although some form of LRS is always present, even if it is only the null version,

the UDI exists only in applications involving an operating system. Intel supply a single UDI for use with the ISIS operating system used on the Intellec development system, in order to allow Pascal-86 programs to execute on the Intellec. If the target system contains an operating system then the user will write a new UDI, and make use of the Intel supplied LRS referred to above. If however the target environment does not include an operating system then it is usual to replace the LRS rather than the UDI, so reducing the run-time system to only two sections.

7.3.1.4. The user written LRS. As a result of the above consideration it was decided to write a LRS to install the Pascal-86 run-time system on the SABUS kit to be used as the tape controller. It was decided to give the LRS the following characteristics :

7.3.1.4.1. I/O devices. The use of a serial terminal as I/O device for local-mode control of the tape controller was discussed earlier. In order to implement this, it was decided to install this terminal as the default I/O device for the run-time system. This means that the terminal will behave precisely as would the screen/keyboard combination on the host system, so allowing program modules to be tested on the development system very easily. It was decided not to install either the tape drives or the data link to the host as I/O devices both because the run time system allows only a limited number of I/O functions which have fixed interpretations, and because both the data link and the tape transports require relatively sophisticated error recovery, to which the LRS environment is poorly suited.

7.3.1.4.2. Error messages. It was decided to route all run-time error messages to the terminal, once more increasing the similarity between development work on the host system and on the target system.

7.3.2. Modularity. A study of the values which K_e takes on in the COCOMO models given in equation 7.1 shows that the larger a program, the larger the amount of effort required to produce a single line of code. The larger the factor K_e in equation 7.1, the more significant this diseconomy of scale becomes. Values of K_e for other published models vary considerably (from 0.91 (economy of scale) to 1.81) but the majority of models (all except for two) show similar diseconomys of scale.

This diseconomy of scale comes about for several reasons, the most significant of which is that of interaction. Consider for example an entity with N objects in it. Between these N objects there are $N(N-1)/2$ possible paths of communication. Although programs do not show anywhere near as great a level of interaction as this it can be said that the more code there is, the more interrelationships there are for the programmer to keep track of, and hence the more difficult the program is to understand and hence to write. This immediately suggests that by minimizing interaction, we can "beat the system", and so minimize programming effort. Consider for example the case of 8 objects in one module (28 possible paths of communication) versus 8 objects evenly split between two modules (12 internal paths of communication plus a certain number of inter-module paths). This can be achieved by programming in modular form, and by application of the technique of information hiding[20]. Information hiding, an extremely important concept in modern software engineering, can be described as the process of limiting as much information as possible to the smallest number of modules possible. This has two major advantages :

7.3.2.1. The interface between modules will be as simple as possible, and so the programmer will only need to know this definition, and the contents of the module on which he is working.

7.3.2.2. Information on the hardware can be limited to only one module. If this is done then in the event of modification to the hardware only one module will need to be changed.

7.3.3. Structure. As may be expected the goal of information hiding can come into conflict with the goal of writing as much as possible of the code in a high level language and the goal of small module size. In this case it was decided to keep all information on the nature of each hardware subsystem in an individual assembler module, and write all other modules in Pascal-86. Further to this it was also decided to split the various layers of the data communication protocol into separate modules. This resulted in a system consisting of seven distinct modules, structured in a loose hierarchy. Of these modules six are user written, while one is the Pascal run-time library as supplied by Intel. This structure is shown in figure 7.1., and consists of the following modules :

7.3.3.1. Initialization module. This is by far the shortest of the modules, consisting only of the code required to initialize the 80188 CPU chip.

7.3.3.2. Main module. This module, written in Pascal, is the main program. It has responsibility for implementing the local operations mode and for the translation of command blocks from the host system into low level tape motion commands.

7.3.3.3. Tape control module. This module, written in assembler, translates the low level tape motion commands generated by the main module into I/O interaction with the timing module hardware.

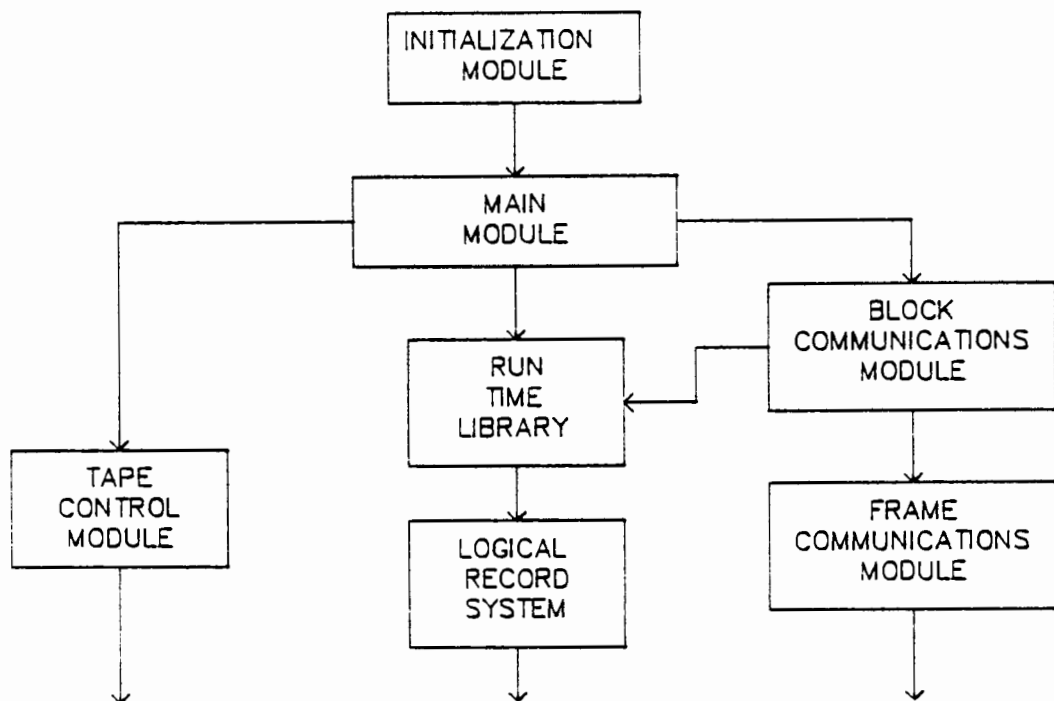


Figure 7.1.

7.3.3.4. Block communications module. This module, written in Pascal, implements the block layer of the communications protocol. It function by passing blocks to and from the main module, and frames to and from the frame communications module.

7.3.3.5. Frame communications module. This module is the lowest level in the data communications subsystem, and is involved with the control of the physical layer (the serial line). It is written in assembler.

7.3.3.6. Pascal run-time library. This is supplied by Intel, and provides language specific run-time support. The existence of this module is essentially transparent to the user, and for most purposes it can simply be regarded as part of the Pascal modules. Note however that this module remains constant in size regardless of the size of the Pascal modules which it serves.

7.3.3.7. The LRS. This module, written in assembler, provides the run-time environment for Pascal-86, as discussed above.

7.3.4. Memory Model. Pascal-86 allows programs to be compiled under one of four different memory models, each of which are suited to different environments. These are the small, medium, compact and large models. The choice of memory models is highly technical, and is significant only in that it defines the length of pointers to data and whether short or long procedure calls are used. It is only necessary to point out that in this case the compact model was used, as it provides compact code, but assembler modules used to interface to compact Pascal modules can still be easily executed on the Intellec development system (this is not the case for the small and medium models). Memory models are extensively discussed in reference [21].

7.3.5. Use of software interrupts. In order to preserve the hierarchical structure of the software, the only form of communication allowed from a lower level to an upper level is a software interrupt. This interrupt informs the higher level

module that some significant event has occurred, and that a call should be executed to the module which generated the interrupt.

7.4. Detailed software description

7.4.1. The initialization module.

The address decoding of the CPU card is configured by registers internal to the 80188 microprocessor. As a result it is necessary to initialize these registers before any other code is executed. This module resides at the microprocessor's reset vector, and so execution begins with this module whenever the tape controller is powered up or is reset. The module simply consists of a string of word output operations, followed by a jump to the start of the main module.

7.4.2. The Logical Record System (LRS).

Only those features of the LRS which are actually used in this application will be described here. Further information may be obtained from reference [21], [22]. The LRS consists of several procedures which are called by the run-time library, in addition to a certain number of data tables. These procedures may be divided into four sections. These are control procedures, I/O procedures, Exception handler procedures and memory management procedures.

7.4.2.1. Control procedures. Control procedures, of which there are three, are involved with the initialization and orderly shut-down of the LRS. These procedures are the following :

7.4.2.1.1. TQINITIALIZE. This procedure is called in order to initialize the LRS, as well as to set the address of the default exception handler. In this case all the data area status flags are set to indicate that the data areas are unused, and the address of the exception handler is set by a call to TQSETERH. Both the function of the data areas and the exception handler will be described in a later section.

7.4.2.1.2. TQGETPRECON. This procedure is called to set connections between a name used in the program and a physical file. As this is not required in this application this procedure is not used.

7.4.2.1.3. TQEXIT. This procedure is called when a Pascal program encounters an END statement. In this case a message indicating that this has occurred is sent to the terminal, followed by a jump to the reset vector of the CPU.

7.4.2.2. Input/Output support. All Pascal-86 I/O operations are directed to logical files, which may in fact be either data files controlled by an operating system or a physical I/O device. I/O support consists of a set of device driver procedures for each file, procedures to connect logical files to a set of device drivers and procedures to create data structures used by the run-time system to control I/O operations. I/O support for the LRS is considerably different to that used for most other systems in that rather than a single device driver per device, Intel make use of a set of device driver procedures. Each of these procedures performs a specific I/O operation. This set of device driver procedures consists of twelve procedures of which one is currently unused. Note that one set of device driver

procedures can support more than one file. For each file name encountered in a program, the run-time system expects to be provided with a pointer to a table which contains the addresses of these twelve procedures. The run-time system hence maintains only a single pointer per file, and when necessary executes an indirect call to the procedure which executes the requires function. It is the responsibility of the user to supply both the set of procedures and the table of addresses of these procedures. These procedure form part of the LRS. Note that a set of procedure must be supplied for each file used by the program, but that the same set may be used for many files.

7.4.2.2.1. Data structures. Two data structures are used by the run time system :

1. File device descriptors. A block of memory called a file descriptor is required to store attributes of an active file. Each file descriptor is 48 bytes long, of which the first 16 are available for use by the device driver associated with the file. This memory space is allocated by the TQFILEDESCRIPTOR procedure to be described later. It is in this space that the address of the table of device driver procedures, among other information, is stored. Note that no information is inserted into the descriptor block by any user supplied code.

2. Device driver tables. A device driver table is an array of pointers to the entry points of the twelve device driver procedures required for each device supported by the run-time system. The function of each individual device driver will be

discussed in a later section. Note that unlike the file descriptor block, the user must initialize this table, which is only read by the run-time system.

7.4.2.2.2. Connection procedures. Two procedures are involved with the allocation of memory for file descriptor and the connection of device drivers with files :

1. TQFILEDESCRIPTOR. This procedure is called by the run-time system before each file is opened to supply space for a file descriptor, as described in section 7.4.2.2.1. In the LRS written for the tape controller this routine can supply only two statically declared descriptors, as required for the standard Pascal INPUT and OUTPUT files. Note that both of these files in fact refer to one device, the serial port.

2. TQDEVICE. This procedure determines which set of device drives is to be used for a particular file. The procedure has as parameter the name of the file, and expects the address of the device driver table for the file in question to be returned. As in this case only one set of device drivers exists, this routine always returns the address of a table containing the address of the device driver procedures associated with the serial port. Note that this is an instance of two distinct files (INPUT and OUTPUT) making use of the same set of device driver procedures.

7.4.2.2.3. The device driver procedures. The device driver procedures consist of eleven procedures, each of which perform a specific I/O operation. One set of these drivers exists for each device in the system. In this particular case only seven of the procedures actually perform a useful function. Note that although two files exist, only one set of device driver procedures are used to process all I/O operations.

The Intel run-time system operates on records and files. Records and files may be delimited by whatever form of separator is appropriate for the file or device. In the case of text files the end of record mark is normally a CR or CR-LF pair, as is the case here. The LRS is also expected to maintain a file pointer, which marks the character at which the next I/O operation will begin. This pointer does not necessarily point to the beginning or end of a record. In this case the LRS buffers a line until the CR key on the terminal is pressed, and also allows line editing via the Backspace key. Note that as the only I/O device used is a terminal, no end-of-file is defined, and hence the LRS will never return end-of-file status. Each of the eleven device driver procedures will now be discussed briefly :

1. OPEN. The run-time system calls OPEN before performing any I/O operation of a file. In this case the serial link is initialized, and the input line buffer flushed.
2. CLOSE. This procedure is called to close a file. This call is ignored by the LRS. Note that this procedure is in fact called twice by the run-

time system, as is OPEN. This occurs because the same set of device driver procedures are used to support two files (INPUT and OUTPUT), both of which are opened and closed.

3. READ. This procedure reads a specific number of bytes from a file. The procedure begins by checking to see if there is data in the line buffer. If not a call is done to local procedure Get_line in order to obtain a line of input. The specified number of characters are then passed to the run-time system. If the end of the line is encountered before the required number of characters can be passed then a status return indicating this is returned along with the number of characters actually transferred.

4. WRITE. This procedure is called to write a specific number of characters to the output file. In this case these characters are simply outputted via the local procedure Conout.

5. SEEK. This procedure is called to set the file pointer for applications which make use of random access files, and is not used in this case.

6. SKIP. This procedure is called to move the file pointer to the beginning of the next record. In this case the line buffer is emptied, and if it is already empty then the next line is read in.

7. END RECORD (E_o_r). This procedure is called to write a record end sequence. A CR-LF pair is sent to the terminal.

8. REWIND. This procedure is called to set the file pointer to the beginning of the file, and is not used in this application.

9. BACKSPACE. This procedure is called to set the file pointer to the beginning of the previous record, and is not supported.

10. END FILE (E_o_f). This procedure is called to write an end-of-file mark. No action is taken.

11. GET FILE INFORMATION (File_info). This procedure is called to obtain information on the current file pointer position and the file length. This is not required in this application and dummy data is returned.

7.4.2.3. Exception handler procedures. The Pascal-86 run-time system makes provision for the exception handler to be varied during program execution. Two procedures in the LRS, in addition to the actual exception handler, implement this ability.

7.4.2.3.1. TQSETERH. This procedure is called to establish the new exception handler. The procedure's only parameter is the address of the handler, which is stored for use by TQGETERH. Note that the default handler is set by a call from TQINITIALIZE, and that in

this particular application it is in fact never changed.

7.4.2.3.2. TQGETERH. This procedure is called by the any procedure in the run-time system which has an error to report. TQGETERH supplies the most recent handler address as set by TQSETERH.

7.4.2.3.3. Exception handler. The exception handler written for the LRS simply writes a message to the local mode terminal containing the error code (these are explained in reference [21]) and the address from which the exception handler was called. Note that this address is only valid if the error in question did not corrupt the stack. The corresponding Pascal line number may be found from this address by consulting the locator map file. Note also that these error messages should never appear during normal operation of the tape controller.

7.4.2.4. Memory management procedures. Memory management procedures are used for the Pascal-86 heap (A heap is the standard Pascal dynamic memory area, which is used to support the NEW and DISPOSE functions), as well as to obtain memory for the run-time system (although this is not documented, version 3 of the compiler requires it). Note that Pascal-86 actually handles memory management for the small memory model in a different fashion to the technique described here, which is used for all other memory models. Two memory management procedures are used :

7.4.2.4.1. TQALLOCATE. This procedure is called in order to obtain a block of memory. The version written

supplies a single block as required by the run-time library, and then no further memory.

7.4.2.4.2. TQFREE. This procedure is called in order to free a previously allocated block. No action is taken in this case.

7.4.3. The Tape Control Module.

7.4.3.1. The tape control module consists of a number of procedures which control the tape transport interface. These procedures all have certain features in common :

7.4.3.1.1. All of the procedures are written to be Pascal callable.

7.4.3.1.2. The operation of the procedures, and specifically the choice of status flags, have been chosen to be as close as possible to that of the DEC TM-11 tape controller.

7.4.3.1.3. None of the command procedures return status information directly. Status returns for all operations are obtained by calling procedure Cntrl_status, which returns the status flags as set by the last operation performed.

7.4.3.1.4. Operations which result in data transfer make use of two buffers. One of these, which contains the data, is supplied by the calling program. The address of this buffer is a parameter of all procedures which make use of data transfer. The other buffer, which contains parity and strobe information on write operations, and parity and state information on read operations, is local to the tape control module. The supplied buffer must always be 8 bytes longer

than the actual data length, in order to allow for check characters which are inserted into the buffer by the tape control module.

7.4.3.2. The module may best be described in terms of the procedures which it contains :

7.4.3.2.1. Procedure Init_tape. This procedure is called in order to initialize the controller module. It's main function is to set the modes of the programmable timer chips in the timing module, but it also checks that the module is disabled. Init_tape is normally only called once only, prior to other procedure.

7.4.3.2.2. Select. This procedure is called in order to set the physical unit number of the tape transport to which subsequent operations refer, as well as to set the timing parameters for the tape transport in question. Once this has been done the procedure need not be called until the user wishes to operate on another unit.

7.4.3.2.3. Erase_tape. This procedure is called in order to erase a tape from the present tape position to the EOT tab. It operates simply by setting the timing module into write mode, and then setting the tape in motion. Once the EOT tab is detected then the motion command is removed. This procedure is an exception among the procedures which cause tape motion in that no data transfer takes place, and the programmable timers play no part in the operation.

7.4.3.2.4. Read_block. This procedure is called to read a block of data from tape. The procedure begins by calculating the settings for the DMA registers, sets these, and then

executes the operation. Once the operation completes, the procedure checks for hardware errors, and then checks that no greater than the requested number of characters were transferred. If no errors are detected the procedure then loops through all the characters read in, checking each one for parity, and calculating the check characters. These check characters are then compared to those read in, and the appropriate status flags set. The procedure then resets the timing module hardware and returns to the caller.

7.4.3.2.5. Write_block. Procedure write block is called in order to to write a block of data to tape. Firstly the programmable timers are set up, and then the parity, strobe enable and check characters are inserted into the buffers by local procedure Fill. Following this a call is executed to procedure Wr_prim, which actually executes the write.

7.4.3.2.6. Write_with_long_gap. This procedure is called to write a block with a long inter-record gap. Execution proceeds exactly as for procedure Write_block except that the timer value for a long gap is used to set up the programmable timer, rather than the normal setting.

7.4.3.2.7. Write_EOF. This procedure is called in order to write an EOF mark. A this procedure is very similar to procedure Write_block, except that dummy data is set up in a local buffer, and no call is done to procedure Fill. The basic Write operation is however still performed by local procedure Wr_prim.

7.4.3.2.8. Space. procedure space is called in order to space the tape by a certain number of blocks. Operation proceeds essentially by setting up the counters, executing the operation and then calculating the number of blocks

actually spaced. This last step is necessary as the space operation will be terminated on encountering either a tape mark, the BOT tab or in the event of a time-out. Note that space operations of only one block are treated as block reads rather than space operations. This is necessary as the counters used in the hardware cannot count only one transition.

7.4.3.2.9. Cntrl_status. Procedure Cntrl_status is called in order to obtain the most most recent tape controller status. The procedure executes by simply checking the hardware status, updating the status bits and returning the results to the caller.

7.4.3.2.10. Off_line. This procedure is called to set the selected tape transport off-line. This is achieved by strobing the appropriate line on the interface.

7.4.3.2.11. Rewind_tape. This procedure is called in order to rewind the tape on the selected unit. As in the previous procedure, this is achieved by strobing the appropriate line on the interface. Note that this is the only procedure which causes tape motion but does not wait for this motion to cease.

7.4.3.2.12. Set_RTH_high. This procedure is called in order to set the selected tape transport's read threshold high, for read-verify operations. The procedure simply sets the interface line to it's active state.

7.4.3.2.13. Set_RTH_low. This procedure is called to set the read threshold to its normal value, and is almost identical to the previous procedure.

7.4.3.2.14. Diag. This procedure is called in order to supply diagnostics information about a particular character in the read data buffer. Information supplied is the state of the format recognizer at the time that the byte was read, and whether or not the character contained a parity error. Note that this information is only valid if the previous operation was in fact a block read operation.

7.4.3.3. Local procedures. The tape control module contain three local procedures which deserve further explanation :

7.4.3.3.1. Wait_for_end. This procedure is used by all procedures which wait for an operation to cease before returning. This procedure loops until either the tape transport reaches the BOT tab, the run bit in the timing module goes inactive or no activity has been detected from the tape transport for a predefined amount of time. Note that the first and last conditions described are in fact error conditions.

7.4.3.3.2. Fill. Procedure Fill is called in order to insert parity, strobe enable and check character information into both the supplied and local buffers. Operation proceeds by simply looping through all the characters to be written, inserting the parity and strobe enable information into the local buffer and calculating the check characters. The check characters are then inserted into the data buffer, together with dummy characters for spaces and the appropriate strobe enable information in the local buffer.

7.4.3.3.3. Wr_prim. This procedure executes a basic write operation on a block of data which already has its strobe and parity bits and check characters set up. Execution proceeds by setting up the DMA registers and then executing

the operation. Once the write is complete a check is done for hardware errors and then a return is executed.

7.4.4. The frame communications module.

7.4.4.1. The frame communications module is responsible for controlling the physical data link to the host computer. The module receives and transmits data frames to the host system, and is responsible for error detection. Note however that this module is not responsible for error correction, which is handled by the next layer of software (by means of frame retransmission). Both the transmit and receive data functions are implemented by means of interrupt driven routines. In addition to procedures for receiving and transmitting frames this module also contain certain other routines related to the control of the data link. Routines are provided to set the baud rate of serial channels, and to provide a time-out function for the next layer of software. The module consists of five procedures which may be called from higher layers of software, and three interrupt handling procedures.

7.4.4.2. Interface procedures. The frame communication modules interface to other modules is via five procedures. These will now be described :

7.4.4.2.1. Init_coms. This procedure is called in order to initialize the frame communications module. The interrupt vectors are set, the serial link hardware initialized and the receive buffer emptied. Following this the timer in the 80188 CPU used to generate clock tick interrupts for the time-out counter is set up. Finally the interrupt controller is initialized, and the interrupts enabled.

7.4.4.2.2. Set_baud. This procedure is called to set the baud rate on either of the two serial channels. The divider ratio to be loaded into the programmable timer used to generate the baud rate is found from a look-up table, and this value loaded into the timer.

7.4.4.2.3. R_frame. Procedure R_frame is called in order to obtain a complete frame which has been assembled by the receive interrupt handler. When a frame has been received the interrupt handler generates a software interrupt to inform the block communications module of this fact. The block communication module then executes a call to this procedure in order to obtain the frame. The procedure copies the frame data and information as to the type of frame received as well as an error flag to the calling module, sets the frame buffer status to empty, reinitializes the check character storage location and then returns to the caller.

7.4.4.2.4. T_frame. This procedure is called in order to transmit a frame to the host system. A check is first done to ascertain whether a frame is already being sent. If so the procedure simply waits until this is complete. The frame to be sent is then copied into local storage, the check character calculation routine initialized and the transmit interrupt enabled. Frame transmission is then completed by the transmit interrupt handler routine.

7.4.4.2.5. Set_timer. The time-out value to be used is simply loaded into local storage. This value will be decremented by the timer interrupt routine to be described later, and an interrupt generated if the count reaches zero.

7.4.4.3. Interrupt handling procedures. The frame communications module makes use of three interrupts, which are central to its operation. These are the following :

7.4.4.3.1. Received character interrupt. This interrupt, which is handled by procedure `Rec_interrupt`, occurs whenever a character has been received by the serial port. The character is checked for transmission errors, and then loaded into a circular buffer. A call to procedure `Asm_frame` is then executed, unless `Asm_frame` is already busy in which case the receive interrupt is enabled and a return executed. Procedure `Asm_frame` assembles characters from the circular buffer into frames. Operation of this procedure is very similar to that of the receiver section of the PDP-11 device driver described earlier with the exception that CRC generation is done via a table look-up technique similar to that described in [23]. The table look-up technique is used in this case as memory usage is not as critical as is the case in the PDP-11. Procedure `Asm_frame` signals the arrival of a complete frame by a software interrupt which will result in a call to procedure `R_frame` as described earlier. Note that procedure `Asm_frame` will continue processing until the circular buffer is empty.

7.4.4.3.2. Transmit interrupt. This interrupt occurs on each occasion that the transmit buffer of the serial port is empty, and is handled by procedure `Tx_int`. Processing is very similar to the transmit interrupt section of the PDP-11 device driver previously discussed. Once processing is complete the transmit interrupt is re-enabled and a return executed.

7.4.4.3.3. Timer tick interrupt. This interrupt occurs on each occasion that a timer tick interrupt occurs (this is

set to 30 Hz by the initialization routine). The interrupt is handled by procedure `Time_int`, which decrements the count if it is positive. If the count reaches zero then a software interrupt is generated to signal to the block communications module that a time-out has occurred. The initial value of the count is set by procedure `Set_timer`, and the timer may be disabled by setting this count value to zero.

7.4.5. The block communications module.

7.4.5.1. The block communications module is responsible for the transmission and reception of blocks of data. Blocks are split into frames, and these are transmitted and received by the frame communications module. Error control is the responsibility of this module and is achieved by means of frame retransmission. The module consists of three procedures which may be called from higher layers of software, and two interrupt handling procedures.

7.4.5.2. Interface procedures. The block communications module contains three procedures which may be called by other modules :

7.4.5.2.1. `Init_block_IO`. This procedure is called in order to initialize the module, as well as the frame communications module. The procedure initializes the frame module, zeros the frame numbers, and sets the interrupt vectors for the two interrupt handler procedures.

7.4.5.2.2. `Block_out`. This procedure is called in order to transmit a block of data. The block to be transmitted is copied into local buffer space, the first frame obtained and then sent. Further frames will be sent only once the current frame is acknowledged. This is done by the frame received interrupt handler, to be described later.

7.4.5.2.3. Block_in. Procedure Block_in is called in order to obtain a received block, and is called in response to a block received interrupt generated by the frame received interrupt handler. The block is simply copied to the calling module, and the block buffer reinitialized.

7.4.5.3. Interrupt handler procedures. The block communications module contains two interrupt handling procedures. Both of these interrupt are generated by the frame communications module.

7.4.5.3.1. Frame received interrupt. This interrupt is generated by the frame module once a complete frame has been received. The frame is fetched, and then the appropriate actions is taken dependant on the frame type. If an ACK frame is received and is correct the next frame is sent. If the frame contains an error, or a NAK frame is received the previous frame is repeated. If a data frame (START character) is received then if it is correct it is added to the block buffer and a ACK frame returned. If the data frame contains an error a NAK frame is returned.

7.4.5.3.2. Time_out. This interrupt occurs when the timer reaches zero as a result of the host not acknowledging a frame. This simply results in the previous data frame being repeated. Note that there is no limit on the number of times that the frame may be repeated as the tape controller can take no useful action in the event of link failure.

7.4.6. The Main Module.

7.4.6.1. The main module is the heart of the tape controller software, and performs two distinct functions :

7.4.6.1.1. Implementation of the local operations mode. This mode, which was described earlier, allows the user to perform various tape operation without host system intervention. This is done by means of commands which are entered interactively at a local terminal. This mode of operation is entirely menu driven in order to make its use as simple as possible for the casual user. The operation and capabilities of the tape controller in this mode are described in full in appendix A.

7.4.6.1.2. Operation in slave mode. The main module is also responsible for decoding and executing command sent to the tape controller by the host system. The procedure responsible for this function is designed to emulate the TM-11 tape controller, as was discussed earlier. This procedure executes in response to a block received interrupt generated by the block communications module.

7.6.2. The two functions described above were included into the same module because they share a number of data structures and procedures as well as having a number of features in common. The module can best be described in terms of the main program and the interrupt handler.

7.6.2.1. Main program. The main program begins execution once the initialization module described in section 7.4.1. has completed. The default timing parameters of the tape drives are set up, as is the logical to physical unit number correspondence. The address vector for the block received interrupt is then set up and the various other modules initialized. A message is then sent to the local terminal informing the user that to enter local mode the enter key on the terminal should be pressed. The main program then waits for this to occur, or a block received interrupt to occur.

Block received interrupt are handled by procedure `Block_rec`, which will be described later.

If the user presses the enter key on the terminal then the tape controller enters local operations mode. Interrupts are first disabled to ensure that blocks from the host will not disrupt the local operations. The user is then prompted with a variety of menus as described in appendix A. Once the user has completed his operations (choice 6 of the main menu) the interrupt are re-enabled and the controller once more waits for the user to request local mode or for a block received interrupt. Space does not allow a complete description of all the procedures used in this module, but several of the most important deserve discussion. These are the following :

7.6.2.1.1. Menu. This procedure is called in order to display a menu and obtain a valid response from the user. Parameters for this procedure are the following :

1. A procedure which displays the menu. This procedure is called by `Menu` in order to provide the screen display and to inform `Menu` as to how many valid menu choices exist.
2. A string which is displayed at the bottom of the menu, an integer which is displayed after this string and a flag to enable or disable the display of the integer. The string is normally used to display error information, or information relating to the previous choice. Note that this string is only displayed on the first occasion on which the menu is displayed. If the menu must be repeated because of an invalid choice on the part of the user then an error message is displayed instead.

Operation proceeds as follows. The parameter procedure is called, and the string and integer displayed. The users response is obtained by procedure `Get_int_in_range`, which will be described later. If this selection is valid then the procedure returns this selection to the caller, and if not the process is repeated until a valid response is obtained.

7.6.2.1.2. `In_int`. This procedure, which is the most complex of all the procedures in the main module, is used to read in an integer in a variety of formats. Integers can be read in either hex, octal, binary, decimal or in the form of a character literal. Protection is provided against invalid characters and overflows. All characters in the number are first read in. If no characters are input then the the variable will not be modified, so allowing the easy implementation of default values. If the number is input as a character literal then this is converted to a number, and this value returned to the caller. If a character literal is not input then the base of the number input is found from the last character in the string, and the number converted character by character. This value is then returned to the caller. This procedure forms the heart of several other procedures, notably the following :

1. `Get_int_in_range`. This procedure is very similar to `Get_int`, but takes extra parameters indicting a valid range for the number, and returns a flag indicating whether the number fell into this range.

2. `Get_char_in_range`. This procedure is very similar to the previous procedure, but is designed to input a character rather than an integer.

3. `Get_value`. This procedure is called in order to obtain an integer in a specific range. It is very similar to `Get_int_in_range`, but continues to prompt the user until a valid integer is input.

7.6.2.1.3. `Clr_screen`. This procedure is called in order to clear the terminal screen before data is sent to it. Control codes for three of the most common terminal types are provided. Provision is also made for terminals which do not make use of control codes, or make use of unsupported control codes.

7.6.2.1.4. `Write_ver`. This procedure is called in order to execute a verified write operation on the selected tape unit. A write operation is done, and the block is then read back. If an error is detected then the operation is retried for the maximum number of retries set. If this is not successful then a gap of 3 inches is left and the operation is again attempted.

7.6.2.1.5. `Wipe_tape`. This procedure is called in order to completely erase a tape. The user is prompted to mount the tape, it is rewound and then erased to the EOT tab, and then rewound again.

7.6.2.1.6. `Tape_copy`. This procedure is called to copy a tape from one unit to the other. The user is prompted to mount both tapes, both tapes are rewound, and then a copy is performed block by block. The copy ends when

either a a double tape mark is encountered for unformatted tapes, and when a end-of-tape block is encountered for an ANSI formatted tape.

7.6.2.2. Command block processor. The command block processor is initiated by a block received interrupt from the block communications module. The block is then obtained from the block communications module and the command, unit number and status bits extracted from it. The codes used for this purpose are defined in appendix D. The unit in question is then selected, and it's status is obtained. The requested command is then executed by means of a case statement. Once the command has been executed the resulting status and return code is inserted into the output buffer and procedure Block_out called in order to transfer the response block to the host.

7.6. Software validation.

Software validation was approached in a modular fashion, and will be described module by module. For validation a bottom-up process was followed.

7.6.1. The LRS. Validation of the LRS was achieved by writing a variety of simple Pascal programs, each of which exercised some aspect of the run-time system. As these programs could also be linked with the normal Intel run-time environment and run on the development system, the results obtained on the tape controller hardware could be compared to "known good" results.

7.6.2. Tape control module. Testing of the tape control module proceeded in tandem with the local operation section

of the main program as well as the final stages of the hardware testing. Testing proceeded in several stages :

7.6.2.1. For the initial testing the basic operation of the timing generation section of the program were investigated. The resulting output waveforms were checked on a logic analyzer. In this way it could be verified that the controller produced write waveforms that matched the tape transport manufacture's recommendations.

7.6.2.2. The second step in the testing involved the test of the format recognition module. For this purpose the controller was looped back. It will be recalled from earlier chapters that the outputs and inputs of the timing module are designed for open collector operation, and that the operation of the format recognition module is dependant only on the read strobe from the tape transports. It is hence possible to jumper the write strobe outputs to the read strobe input of the timing module, and simulate the operation of a tape drive. In this way the operation of the format recognition module could be tested under a wide variety of simulated conditions.

7.6.2.3. Once the module had been tested under simulated conditions the tape transports were connected to the tape controller, and the controller's ability to write and read back tapes was verified. The next step involved tests to ensure that the controller could in fact read and write industry standard tapes. Once this was verified the tape control module as well as the hardware could be assumed to be functional.

7.6.3. The communications system. The communications system, which consists of the frames and block communications modules, was tested in a very similar fashion to the PDP-11 device driver's communication functions. The frame module was first tested (in conjunction with a dummy block module) with frames generated on a terminal, as well as in loop-back. Once the basic operation of this module was verified the block module was tested, once more by means of loop-back testing. Note that this testing was possible because of the use of local storage for received and transmitted blocks, and the interrupt driven nature of the software.

The final step in testing the communications system was to write a dummy main module which simply returned fatal hardware error codes, regardless of the contents of the incoming block. With this module in place the tape controller could be connected to the PDP-11, and the operation of the link verified.

7.6.4. Main module. Testing of the main module involved testing both the local operations mode and the slave controller mode. Testing of this module also involved testing the system as a whole.

7.6.4.1. Testing the local operations mode. Most of the user interface code (the menu software, numeric input code etc.) was tested on the development system before transfer to the controller. Once this software was operating satisfactorily the tape controller module was integrated into the system, and tested as described above. Once this was complete the code concerned with the local operations mode of operation had been totally tested.

7.6.4.2. Testing the system under host control. Once the controller had been tested for local mode operation the communications modules were integrated into the system and the TM-11 emulation capabilities of the software tested. This proceeded in several steps :

7.6.4.2.1. Initially the system was tested with the RTL program described in the chapter on the device driver. This allowed the the test of the controller for all possible operations, and also allowed the controller's responses to be compared to that of the Cypher tape drive discussed earlier. As this drive is a commercial hardware emulation of the TM-11 tape system compatibility with the TM-11 could be verified.

7.6.4.2.2. The next step in the process involved the testing of the system in conjunction with the PDP-11's operating system software. RSX-11, the operating system in question, allows tape drives to be treated exactly as any other mass storage device (such as a disk drive). In DEC's words the TM-11 is "mountable as a Files-11 device". This means in essence that the normal operating system commands used for file manipulation can be used on the tape drive. The user can for example obtain a directory by using a standard command, mount and dismount a unit, assign a unit for use only by himself or obtain status information from the device. This is the most critical area of compatibility, as if the tape controller is compatible with the operating system then all "well behaved" programs (programs which use the I/O hierarchy as discussed in an earlier chapter)

will execute correctly. Testing this capability involved executing the various operating system commands which are legal for tape drives, and verifying that the expected results occurred.

7.6.4.2.3. In addition the the operating system, which has imbedded in it the commands discussed above, DEC also supply certain utility programs. Although these programs do not form a part of the operating system, and often are designed for use only with certain peripheral devices, the tape controller should be able to operate in conjunction with them. Three utility programs supplied by DEC can operate in conjunction with tape drives :

1. Peripheral Interchange Program (PIP). PIP is supplied in order to allow the manipulation of files on all Files-11 devices. The user may copy files from device to device, rename files, delete files etc. As this utility is designed for use on all Files-11 devices the tape controller should be able to operate in conjunction with it, and indeed testing showed that this was the case.

2. File Transfer Utility (FLX). FLX is design to allow the transfer of Files-11 format files to and from media which is not Files-11 formatted, and according to the manual operates only in conjunction with certain devices (including the TM-11). Testing showed that FLX rejected commands which referred any

device not listed as compatible with it, including MA-drive. Accordingly MA-drive was modified to be called MM-drive, which is a listed device but which is not used at UCT. This functioned correctly, but was removed from the system as this would be incompatible with a system which included a true MM-drive, and would be very bad practice, as it renders a standard device unworkable.

3. Backup and Restore Utility (BRU). BRU is provided to simplify the task of backing up disks to tape. As is the case with FLX, BRU is specified to operate only in conjunction with certain devices. When tested BRU accepted all commands, and apparently executed them without error. However when a verify pass was executed it was discovered that incorrect data was written in certain tape blocks. Although a considerable amount of time was expended in investigating this problem no solution could be found. The problem was traced to BRU corrupting its own data blocks on write operations, apparently with data from subsequent blocks. This corruption was not repeatable, and only appeared after several data blocks had already been written correctly.

The problem is apparently related to the storage of the address of the data block, and its manipulation by the device driver. It will be recalled that an inherent difference between the TM-11 controller and the device

driver written was that the TM-11 makes use of DMA, and that the storage of the block address for DMA devices is different to that for program transfer devices. Accordingly the device driver was modified to accept addresses in DMA device format, and then convert them to a format usable by the executive get and put byte routines. This did not solve the problem, although it reduced the frequency of corrupted blocks considerably. The local agents for the system, when approached on the problem, stated that they were not aware of any such problem on any DEC supported device, but that "BRU is a mass of errors". As operation in conjunction with BRU is not necessary, nor does the BRU manual suggest that this is possible, the search for a solution was abandoned. In all other respects however the tape controller performed flawlessly.

CHAPTER 8

CONCLUSION

8.1. In conclusion we will first examine to what extent this project has achieved its aims, and then discuss some of the more interesting lessons to be learned from the execution of this project.

8.2. Goals of the project

The goals of this project were defined in the first chapter, and essentially amounted to the construction of a magnetic tape controller suitable for use at a location remote from the host system. This has been achieved. The controller as constructed interfaces successfully to the PDP-11 system, and because of its modular structure should be easy to modify for use in other applications. The tape controller can also be very easily interfaced without modification to any computer system equipped with a RS-232 port. For example, an IBM-PC could easily control the tape drives, allowing tapes for machine control applications to be written.

8.3. System Performance

As may be expected the performance of the tape controller units in terms of speed is totally overshadowed by the transfer time of the serial link. This transfers approximately 1000 characters per second, a rate considerably below that of conventional mass storage devices. As was explained in chapter 1, it is intended that the controller should be modified to allow operation on a local area network, which will increase this transfer rate by several orders of magnitude. Special provision was made in both hardware and software to ease this modification. Until the LAN is implemented however the transfer rate of the controller significantly reduces the usefulness of the controller. Two techniques were considered to improve the transfer rate of the controller as it stands. These were the following :

8.3.1. Increase the baud rate of the serial channel from its current 9600 to 19200 baud. This may be done relatively simply, and doubles the transfer rate, but requires hardware modification to the PDP-11 serial interface. A suggestion to this effect was not greeted with enthusiasm by the system manager, and hence this was not pursued.

8.3.2. Some form of data compression could be used on the serial link. This was rejected, as a rough analysis of data transferred on the link suggested that the increase in transfer rate would be less than 5%, at the expense of a considerable increase in the size and complexity of the PDP-11 device driver.

8.4. Software Engineering Aspects

The structure of the tape controller software was chosen in the previous chapter in order to minimize programming effort. This was justified by reference to the basic COCOMO model, which predicts programming effort in terms of lines of code. It is instructive to now calculate the theoretical effort based on this model. In this case the semidetatched model of software development is most appropriate. In total the tape controller software (note that this excludes the device driver) consists of approximately 3200 lines of software, implying a nominal development time of 11 man-months. If the software is assumed to be developed in embedded mode then this figure rises to 15 man-months. The actual figure, as estimated by the author, was approximately 4 man-months. In fact these figures are less significant than may be thought at first sight. Firstly, it is difficult to compare figures from a model developed for commercial use to figures obtained for an academic project, and secondly the basic model used to obtain these figures is not particularly accurate. Considerably more accurate forms of the COCOMO model exist, but are difficult to apply in this case, as they would require the author to assess his own competence as a programmer, a task the author declines. It can however be said that the effort was at least certainly not greater than average, and probably considerably less. In the author's opinion however a more significant factor in this lower than average effort was the installation of the Pascal run-time system in the target system.

8.5. Installation of the Pascal run-time system.

The installation of the Pascal run-time system was explained in the previous chapter, and the advantages of this discussed at length. Of necessity an evaluation of its usefulness must be

subjective, but a single example will serve to illustrate the value of the run-time system. Approximately two thirds of the way through this project a new version of the Pascal compiler was installed on the Intellec development system. Needless to say, the author's program immediately ceased to operate on the target system. This was in fact as a result of two flaws in the Pascal-86 compiler. The first of these resulted in the compiler generating invalid object records (this is described more fully in appendix H) and was fairly easy to find as absolutely no code functioned. This meant that the problem could be found from a very simple program which did not make use of the Pascal library (in fact a program consisting of a single instruction!). The second problem could however have been considerable more complex, and involved the program failing only when run-time library functions were used. The problem was in fact an undocumented call to the memory allocation routines, which resulted in an immediate error if no memory was available. In this case the cause of the problem was immediately evident as the error handler provided an error message with a status code signifying a memory allocation error. If however no run-time error handling was available the tape controller would simply have crashed, for no explicable reason, immediately the program was run. The author would then have been faced with a program which could run on the development system, but simply failed without providing any output on the target system. Although this is not a common fault, without the installation of the run-time system it would have caused considerable trouble. In the author's opinion the installation of the LRS more than repaid the effort involved in it, considerably aiding in the debugging of the tape controller software.

REFERENCES

1. IBM 2400-Series Magnetic Tape Units Original Equipment Manufacturers' Information, IBM Form 226862-4.
2. "USA Standard Recorded Magnetic Tape for Information Interchange (800 CPI, NRZl)", United States of America Standards Institute.
3. Siewiorek, D.P. and Swarz, R.S.: "The Theory and Practice of Reliable System Design", Digital Press, 1982.
4. Lin, S. and Costello, D.J. Jr: "Error Control Coding : Fundamentals and Applications", Prentice-Hall, 1983.
5. Wiggert, D: "Error Control Coding and Applications", Artech House, 1981.
6. "Mod 10 NRZI Magnetic Tape Transport : Operation and Maintenance Manual", Wanco, Publication 200237 P, June 1976.
7. "Model 6X60 Synchronous Write Synchronous Read Tape Transport", Operating and Service Manual No. 101133, Pertec, 1971.

8. "Recommended Microprocessor Bus Standard For South Africa", National Electrical Engineering Research Institute, NERI/Ko/1/78, Issue 2, 7 May 1979.

9. "SABUS - the SA uP bus standard", Pulse October 1984.

10. "iAPX 188 High Intergration 8-bit Microprocessor", Intel Data Sheet 210706-001, Intel Corporation, 1982.

11. "Introduction to the 80186", Intel Application Note, contained in Microsystem Components Handbook, Intel Corporation, 1984.

12. "iAPX 86/88, 186/188 User's Manual", Vol.1 Programmer's Reference, Intel Corporation, May 1983.

13. Neubig, B: "Technical Information : Design of Crystal Oscillator Circuits", Special Issue of VHF Communications, Edition 3/1979 and 4/1979.

14. Zimmermann, H: "OSI Reference Model - The ISO Model for Open Systems Interconnection", IEEE Trans. Commun., vol COM-28, April 1980.

15. Tanenbaum, A.S.: "Computer Networks", Prentice-Hall, 1981.

16. "RSX-11M Version 4 Executive Reference Manual", Digital Equipment Corporation, 1980.

17. "IAS/RSX-11 I/O Operations Reference Manual", Digital Equipment Corporation, 1981.

18. "RSX-11M Guide to Writing an I/O Driver", Digital Equipment Corporation.
19. Boehm, B.W.: "Software Engineering Economics", Prentice-Hall, 1981.
20. Parnas, D.L.: "Designing Software for Ease of Extension and Contraction", IEEE Trans. Software Engineering, March 1979.
21. "Pascal-86 User's Guide", Intel Corporation, 1983.
22. "Run-Time Support Manual for iAPX 86,88 Applications", Intel Corporation, 1982.
23. Perez, A.: "Byte-Wise CRC Calculations", IEEE Micro, V3 N3, June 1983.
24. "Kennedy Model 9700 Magnetic Tape Unit Operating Manual", Kennedy Corporation, Publication number 106-9701-700A.
25. "Argus 500 Computer System Hardware Technical Manual (Cover 3, Part 4)", Ferranti Limited, 1979.
26. "Magnetic Tape Coupler Model T04/C", Dataram Corporation, Publication number 06133.
27. "MECL High Speed Integrated Circuits", Motorola inc., 1978.

APPENDIX A

TAPE CONTROLLER USER'S GUIDE

This appendix contains information required to operate the tape controller both as a peripheral of the PDP-11 computer, and as a stand-alone device. The appendix consists of four sections, the first of which contains general information on the controller, and the second of which contains information required to load the PDP-11 device driver onto the PDP-11. This information is only required by the system operator. The third section contains information on the use of the tape controller as a peripheral of the PDP-11 for the general user. The fourth and last section describes the operation of the controller as a stand-alone device. Knowledge of this section is not required to operate the controller as a PDP-11 peripheral.

NOTE. Information presented in this appendix is correct as at 23 June 1986.

1. General Information

1.1. The tape controller can operate in one of two modes: remote mode (controlled by a host system) or local mode (controlled by the user on a local terminal). Operation in these two mode is mutually exclusive, in order to prevent access conflicts.

1.2. For both local and remote mode, logical unit numbers are used. The upper unit in the rack is unit 0, and the lower unit is unit 1.

1.3. For any tape operations to occur the unit in question must be loaded and on-line. Procedures for achieving this vary from unit to unit. Consult the relevant operating manual for this and other information specific to each individual tape transport.

1.4. As currently configured, the tape controller software is in the form of a hex file which must be down loaded before any other operation is performed. This may be most easily achieved by making use of the PDP-11 serial channel used for data communication. Note however that this is only possible before the channel has been reconfigured to operate as a data link. The physical connection to achieve this is exactly the same as for the data link, and is described in section 2. Assuming this connection to be in place, then the following command should be used :

```
PIP TT2: = DL1:[100,34]admtcn.hex
```

2. Physical Interface

2.1. Interface to the tape transports. The following connectors are used for interface to the tape transports :

Card 1 J4	Write data.
Card 1 J2	Control.
Card 2 J2	Read data.

Card 1 J3 is connected to card 2 J3.

2.2. Serial Interface channels. The serial interface channels both come from the CPU card. They are connected as follows :

Channel 0	Data link to host system. In the current configuration this goes to TT2: on the PDP-11. Note that channel 0 is identified by a red band on the ribbon cable leading to the RS-232 connector.
Channel 1	This is connected to the local terminal. Note that this connection is not required for operation in remote mode.

3. Loading the PDP-11 Device Driver

3.1. Note that if the tape controller software is to be downloaded via the PDP-11, then the down load must be performed before the device driver is loaded.

3.2. The following example contains the complete load sequence, as seen on the user's terminal (user inputs underlined). Note that this must be performed from a privileged terminal. The (ESC) shown below represents a single ESC character.

3.3. Example

```
)ope 350  
000350 /117034 2744  
000352 /000342 0  
000354 /116760 2744  
000356 /000342 0  
000360 /117034 (ESC)  
)loa ma:  
)
```

3.4. Use of a different PDP-11 serial line. As currently configured the device driver can make use only of serial line TT2. In order to use a different serial line on the PDP-11 the following steps must be taken.

3.4.1. The device driver data table must be modified to represent the the new base address of the serial port in question, as well as it's associated interrupt vectors. This information is stored in the SCB.

3.4.2. The sequence shown above must be modified so as to set the interrupt vector of the new serial link to the nonsense vector. Thus the OPE 350 above must become OPE XXX,

where XXX is the interrupt vector of the serial port in question.

4. Using the controller in remote mode

4.1. The device driver is known as MADRV, and the tape transports are MA0: and MA1:.

4.2. Operation of the tape units once the above load sequence has been completed is essentially identical to that of a conventional tape drive. There are however two exception to this :

4.2.1. The utility programs BRU and FLX do not function in conjunction with MADRV. This is discussed in detail in the main text.

4.2.2. MADRV provides an additional error message. If the data link is broken, or is not reliable enough to allow error free transmission, then the error code IE.TMO will be returned. Note however that certain utility programs may also return a device-not-ready message, rather than IE.TMO.

4.3. Example. In the following example a blank tape is initialized with a volume name of test, the tape is mounted as a FILES-11 device, a file copied to tape and finally a directory of the tape obtained. User inputs are underlined.

```
)  
)all mal:  
)ini mal:test
```

```
)mou mal:test  
)pip mal: = [100,34]clean.cmd  
)dir mal:
```

Directory MAL:
20-JUN-86 14:28

CLEAN.CMD ;1 1. 20-JUN-86 00:00

Total of 1./1. blocks in 1. file

)

4.4. Possible error messages.

The PDP-11 may return the following error messages (All numeric values are in octal).

IE.FHE (305)	The unit was not ready, or a catastrophic hardware error occurred, so preventing the operation from being attempted.
IE.BBE (310)	A bad block was encountered, and the error was not recoverable.
IE.ABO (361)	Operation was aborted on the request of the user.
IE.DAO (363)	Data overrun. A block which was read in was larger than the the stated size.
IE.WLK (364)	A write operation was attempted on a unit which was either physically

write-locked, or had the write-lock bit in the status word set.

IE.SPC (372)	Illegal buffer size. A byte count of less than 8 was specified for a read operation, or a byte count of less than 14 for a write operation.
IE.VER (374)	Irrecoverable parity or CRC error.
IE.IFC (376)	Illegal function request.
IE.TMO (241)	Time out on request. This is returned in the event of a data link failure.

5. Operation in local mode

5.1. Information common to all local mode operations :

5.1.1. Local operations mode is an interactive, menu driven system which enables the user to examine, modify and copy tapes without host intervention. Diagnostics information may also be obtained, and various aspects of the operation of the controller configured. Note that it is not necessary for connection to be made to a host system for operation in local mode .

5.1.2. Any numeric value may be entered in any one of five formats :

5.1.2.1. Hexadecimal. Hexadecimal numbers are terminated by an 'h' or 'H' character.

5.1.2.2. Octal. Octal numbers are denoted by termination by one of the following characters : 'o', 'O', 'q' or 'Q'.

5.1.2.3. Binary. Binary numbers are denoted by termination with either a 'b' or a 'B'.

5.1.2.4. Decimal. A decimal number is denoted by a number with no special terminating character (the number ends in a valid decimal digit).

5.1.2.5. Character literal. The ASCII value of a character can be entered as a numeric parameter if it is enclosed by double quotes eg. "t". Note that while all other numeric input formats are insensitive to case, character literal input will preserve case.

5.2. Entering local mode. Local mode is entered by transmitting a carriage return character from the local terminal. The user is then prompted by a series of menus. Each of these menus will now be described.

5.3. Local mode main menu. The main menu displays immediately that local mode is entered, and is shown below :

LOCAL MODE MENU

1. Copy an unformatted tape.
2. Copy an ANSI formatted tape.

3. Erase a tape.
4. System functions.
5. Enter local tape operations mode
6. Return to remote mode.

Input your selection (1..6):

These menu choices have the following functions :

5.3.1. Copy an unformatted tape. The user is prompted to mount a source and destination tape, and then a block by block copy operation is performed. Each block is verified after it has been written. The copy operation halts on encountering a double tape mark, and should be used for tapes which are not ANSI formatted. Once the operation is complete both tapes are rewound. Information as to the number of blocks and tape marks copied are displayed while the copy is in progress.

5.3.2. Copy a formatted tape. This option performs a very similar function to the above option, but copies ANSI formatted tapes rather than unformatted tapes. In this case the copy operation terminates on encountering an tape mark preceded by an EOV block. Note that as currently configured, UCT's PDP-11 system produces ANSI formatted tapes.

5.3.3. Erase a tape. The user is prompted for the unit number on which the tape is mounted. The tape is then rewound, erased to the EOT tab, and then rewound again. Note

that this destroys all information on a tape, and cannot be used to erase from the current tape position to the end of the tape.

5.3.4. System functions. This option allows the user to configure certain aspects of the tape controller's operation. The system functions menu is displayed. This is discussed below.

5.3.5. Local tape operations. This option allows the user to selectively modify the contents of a tape mounted on either unit, as well as to obtain diagnostics information. The local operations menu is displayed. Local operations are discussed below.

5.3.6. Return to remote mode. Once this option has been selected, the controller is once more under the control of the host system. The controller can be returned to local mode by pressing the carriage return key on the local terminal.

5.4. System functions menu. The system functions menu is shown below :

SYSTEM FUNCTIONS MENU

1. Set data link baud rate.
2. Set terminal baud rate.
3. Set terminal type.

4. Set tape unit characteristics.
5. Enable block diagnostics.
6. Disable block diagnostics.
7. Enable frame diagnostics.
8. Disable frame diagnostics.
9. Return to local mode menu.

Input your selection (1..9):

These menu choices have the following functions :

5.4.1. Set data link baud rate. If this option is selected, a menu of baud rates is displayed. This option is provided primarily in case the system is to be used in conjunction with some other host computer, for example via a telephone line.

5.4.2. Set terminal baud rate. This option functions identically to the above option, but sets the terminal baud rate. This is provided for terminals which cannot support the default 9600 baud. This could for example be used to interface to a printing terminal in order to provide hard copy of tape contents.

5.4.3. Set terminal type. Before a menu is displayed the terminal screen is cleared. This option selects the terminal type in use in order select the appropriate control codes.

If the terminal in use is not listed in the displayed menu then the final choice, 'Other', should be used. This outputs no control codes. The default terminal type is Perkin Elmer.

5.4.4. Set tape unit characteristics. This option is provided in order to modify the timing characteristics of a tape transport connected to the controller. The following values are prompted for.

NOTE. These values should be modified only after studying the information pertinent to tape formats presented in the main section of this document.

5.4.4.1. Master clock. This value sets the period of the master character clock. The character clock is the multiple of the tape bit density in bits per inch (always 800) and the tape speed in inches per second. This value is in units of 200 nS clock increments.

5.4.4.2. Short gap count. This value sets the counter which detect the gap in read strobes which occurs between the data characters and the check characters when a block is read. The value chosen must be large enough to prevent the detection of a gap in the event of normal variations in bit density, but must be small enough to always detect the gap in question. This is normally set to two character spaces. This value is in units of 200 nS clock increments.

5.4.4.3. Long gap count. This value sets the counter which is used to detect a gap between read strobes long enough to indicate that the present block (or tape mark) is complete. This is normally set to a least 12

character spaces. This value is in units of 200 nS clock increments.

5.4.4.4. Normal start gap length. This is used to set the time delay from initiating tape motion to writing the first character for write operations. This value is in character clocks, and is obtained from the tape transport's manufacturer.

5.4.4.5. Long start gap length. This is used to set the time delay from initiating tape motion to writing the first character for write-with-long-gap operations. This value is in character clocks, and is obtained from the tape transport's manufacturer.

5.4.4.6. EOF start gap length. This is used to set the time delay from initiating tape motion to writing the first character for write tape mark (EOF) operations. This value is in character clocks, and is obtained from the tape transport's manufacturer.

5.4.4.7. Read start gap length. This is used to set the minimum gap from initiating tape motion to when read characters may be expected. This value is not critical, and is normally set to approximately a quarter of the value used for write operations.

5.4.4.8. Stop gap length. This value sets the gap from the last character to be written for write operations or, in the case of read or space operations, from the detection of the end of the block to the removal of the tape motion command. This, in conjunction with the start gap value, sets the inter-block gap length. This

value is in character clocks, and is obtained from the tape transport's manufacturer.

5.4.5. Enable block diagnostics. Selecting block diagnostics results in the tape controller displaying the contents of the various fields in each command block which it receives. Note that the data field is not displayed, but that the status of the tape unit after the operation has completed is displayed. A description of the fields in command blocks may be found in appendix D. This option is supplied to ease the task of interfacing the tape controller to another host system. Note that the tape controller must be returned to remote mode before it will respond to any incoming blocks.

5.4.6. Disable block diagnostics. The block diagnostics system described above is disabled.

5.4.7. Enable frame diagnostics. Selecting frame diagnostics results in the tape controller displaying the contents of the various fields (other than the data) in each frame which it sends or receives. A description of all possible frame types and fields may be found in appendix D. This option is supplied to ease the task of interfacing the tape controller to another host system.

5.4.8. Disable frame diagnostics. The frame diagnostics system described above is disabled.

5.4.9. Return to local mode menu. The local mode main menu is displayed.

5.5. Local operations menu. The local operations menu is shown below :

LOCAL OPERATIONS MENU

1. Select a unit.
2. Read a block into the local buffer.
3. Write the local buffer to tape.
4. Write to tape with a long inter-record gap.
5. Write an EOF mark.
6. Space blocks forward.
7. Space blocks reverse.
8. Set the read threshold high
9. Set the read threshold low
10. Rewind.
11. Set unit off-line.
12. Display the local buffer.
13. Modify the local buffer.
14. Display local buffer diagnostics information.
15. Change the present buffer size.
16. Exit to the main menu.

Unit 0 status : On line, at BOT

Input your selection (1..16):

5.5.1. All local operations have several features in common :

5.5.1.1. A status line is displayed at the bottom of the local operations menu. The information in this line is updated after each menu selection. The following messages can appear on this line :

5.5.1.1.1. Unit number.

5.5.1.1.2. Either On-line or Select error. If the unit is not on-line, then the select error message will be displayed.

5.5.1.1.3. At BOT. This message is displayed if the selected unit is at the BOT tab.

5.5.1.1.4. Past EOT. If this message is displayed then the unit is past the EOT tab. This is reset once the unit has returned past the EOT tab.

5.5.1.1.5. Rewinding. The unit is rewinding. Note that the next command can be entered while this message is displayed. The command in question will be delayed until the rewind operation is complete.

5.5.1.1.6. Time error. This is a time-out condition. It occurs on read or space operations when no read strobes are detected for approximately 12 inches of tape.

5.5.1.1.7. Parity error. A parity error was detected on a read operation.

5.5.1.1.8. CRC error. The CRC check on a block read in failed.

5.5.1.1.9. Length error. The block to be read in was too large.

5.5.1.1.10. DMA error. This is a hardware error, and can result either from incorrect timing parameters, or a corrupted tape block.

5.5.1.1.11. Write locked. The write enable ring is not installed on the tape mounted on the selected unit.

5.5.1.2. All operation refer to the selected unit. A unit is selected by option 1 in the menu, to be described later.

5.5.1.3. All operations involving data transfer refer to the local buffer. This buffer also has associated with it a length, which is normally set by the most recent read operation, but may be manually changed. On initial entry into the local operations menu the buffer is initialized to contain a test message of length 16.

5.5.2. The following operations may be performed :

5.5.2.1. Select a unit. The user is prompted for a valid unit number. All subsequent operation refer to this unit.

5.5.2.2. Read block. The next block of data on the selected unit is read into the local buffer. Note that the buffer size is set to maximum prior to the read. Note that reading a tape mark will corrupt the contents of the local buffer.

5.5.2.3. Write block. The local buffer is written to tape.

5.5.2.4. Write block with long gap. The local buffer is written to tape with a long inter-record gap.

5.5.2.5. Write EOF. A tape mark is written to tape.

5.5.2.6. Space blocks forward. The user is prompted for the number of blocks to space. The operation is then executed.

Note that the space operation terminates on encountering a tape mark.

5.5.2.7. Space blocks reverse. The same conditions as for the above operation apply.

5.5.2.8. Set the read threshold high. The read threshold of the selected tape transport may be set high in order to verify a previously written block. Note that the normal setting is low.

5.5.2.9. Set the read threshold low.

5.5.2.10. Rewind. The tape on the selected unit is rewound.

5.5.2.11. Set the unit off-line. The unit is set off-line. No further operation can be performed until the tape transport is set on-line by the operator.

5.5.2.12. Display the local buffer. The hex value of the characters, as well as their ASCII representations are displayed. Note that characters which are not printable are denoted by a period ('.').

5.5.2.13. Modify the local buffer. The user is prompted for the location of the first character to modify. Note : Numbering starts at 0. The current value of the character is displayed, and the user is prompted for the modified value. This process then continues for the rest of the buffer, or until the user inputs an invalid value. Note that a carriage return causes the next character to be displayed without changing the current character.

5.5.2.15. Display diagnostics information. This displays the characters in the local buffer, together with the following information :

5.5.2.14.1. Whether or not a parity error occurred when the character in question was read in.

5.5.2.14.2. The state of the format recognition module after the character was read in. This information is important when setting the tape transport in question's timing parameters. The following state values can occur :

5.5.2.14.2.1. State 1. The first character in a block, and no other, should have a state value of 1. This denotes that both the tape controller detected both a long and short gap prior to the character.

5.5.2.14.2.2. State 2. The rest of the data characters should be of state 2, denoting that no gap was detected prior to the character.

5.5.2.14.2.3. State 3. The first check character should have a state value of 3. This denotes that a short gap was detected prior to the character's read strobe. If no error has occurred, then this character is the first check character in the block. Note that this check character is automatically displayed as the final character in the block when displaying diagnostics information.

State values other than these, or these state values at invalid positions in the block, indicate either incorrect timing parameters or a corrupted block. Note that valid diagnostics information can only be obtained directly after a read operation.

5.5.2.15. Change the local buffer size. This option is used to change the length of the local buffer. This is typically required in order to write a block which is either shorter or longer than the current block block length, for diagnostic purposes.

5.5.2.16. Return to main menu. This return to the main local mode menu.

5.5.3. Example. The following example shows the first block on unit 1 being read into local memory (user inputs underlined).

To return to local mode hit return

LOCAL MODE MENU

1. Copy an unformatted tape.
2. Copy an ANSI formatted tape.
3. Erase a tape.
4. System functions.

5. Enter local tape operations mode
6. Return to remote mode.

Input your selection (1..6):5

LOCAL OPERATIONS MENU

1. Select a unit.
2. Read a block into the local buffer.
3. Write the local buffer to tape.
4. Write to tape with a long inter-record gap.
5. Write an EOF mark.
6. Space blocks forward.
7. Space blocks reverse.
8. Set the read threshold high
9. Set the read threshold low
10. Rewind.
11. Set unit off-line.
12. Display the local buffer.
13. Modify the local buffer.
14. Display local buffer diagnostics information.
15. Change the present buffer size.
16. Exit to the main menu.

Unit 1 status : On line, at BOT

Input your selection (1..16):2

LOCAL OPERATIONS MENU

1. Select a unit.
2. Read a block into the local buffer.
3. Write the local buffer to tape.
4. Write to tape with a long inter-record gap.
5. Write an EOF mark.
6. Space blocks forward.
7. Space blocks reverse.
8. Set the read threshold high
9. Set the read threshold low
10. Rewind.
11. Set unit off-line.
12. Display the local buffer.
13. Modify the local buffer.
14. Display local buffer diagnostics information.
15. Change the present buffer size.
16. Exit to the main menu.

Unit 1 status : On line

Read block size : 80

Input your selection (1..16):

APPENDIX B

CIRCUIT DIAGRAMS : CPU CARD

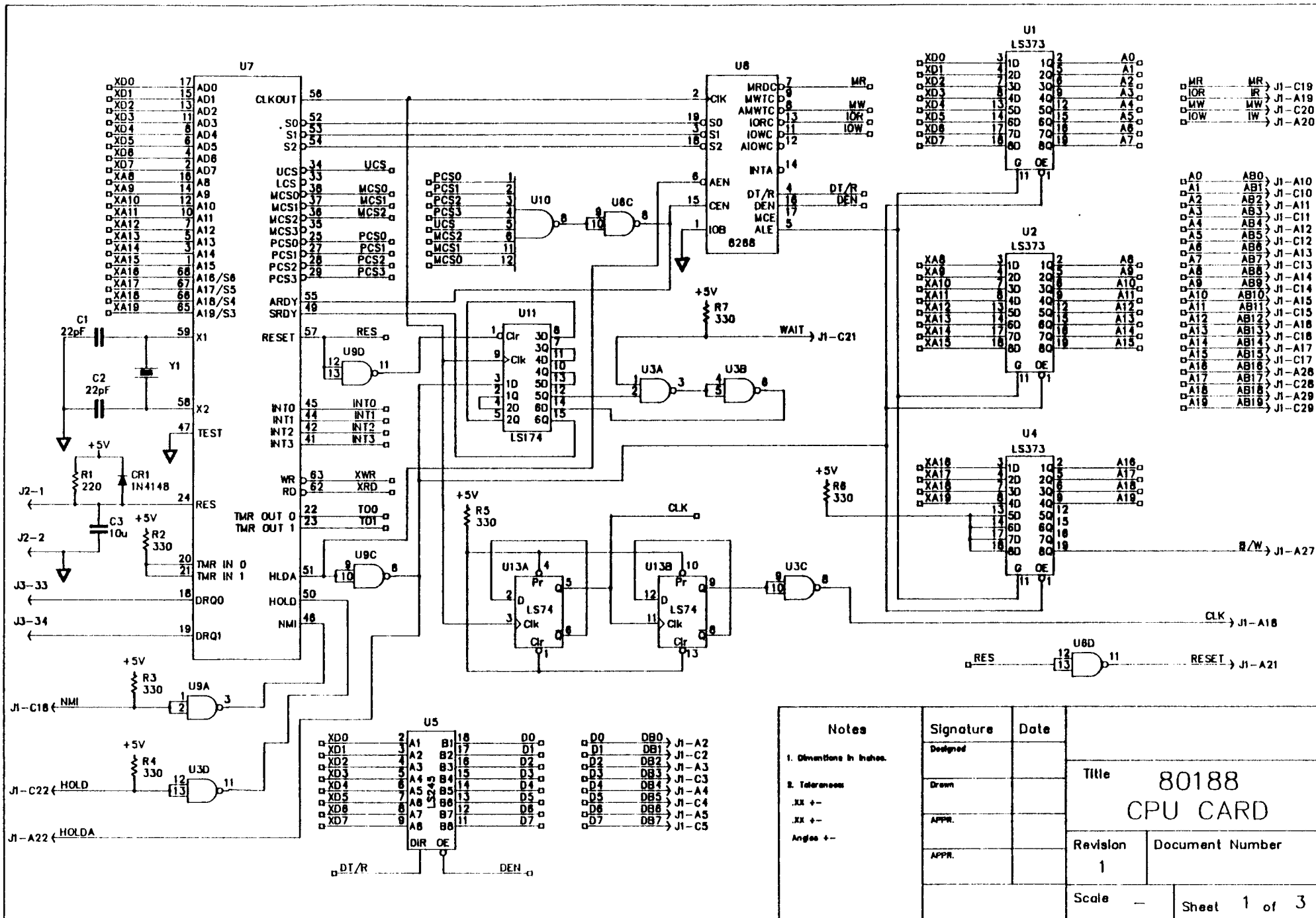
The CPU card consists of the following subsections :

1. CPU. The CPU (U7) is an 80188-6. This is the 6 MHz version of the device. Although an 8 MHz version of the device exists, and would be better suited to this application, this was not available at the time of purchase. The clock to the chip is provided by a 12 MHz crystal and the device's internal oscillator. U11 and its associated circuitry provide synchronization of the ready line from SABUS, as well as ensuring that timing margins are not violated when an external device releases a bus hold.

2. SABUS interface. U1, U2 and U4 latch the CPU address and drive both the local and SABUS address lines. U8 provides control signals to SABUS, and is disabled by U10 in the event of a local bus access. U8's control lines are tristated in the event of a bus hold. U5 drives the SABUS data lines, and is controlled by U8's bus control lines. U13 divides the CPU clock down to a frequency compatible with SABUS cards, and U6d provides drive to the SABUS reset line.

3. EPROM storage. U14, U15, U16 and U17 form the EPROM storage for the card. U15, which contains the boot code, is enabled by the CPU's UCS line. This UCS line is the only chip select line which is active when the CPU is reset. All the other EPROMs are selected by MCS lines.

4. Serial ports. U18 and U21 are the serial interface chips, and U19, U20, U22 and U23 provide the RS-232 conversion function. U18 is selected by PCS0 and U21 by PCS1. The baud rate for serial port 0 is set by the T0 output of the CPU and that of serial port 1 by the T1 output. The receive and transmit interrupts of both ports are connected to individual interrupt inputs on the CPU.



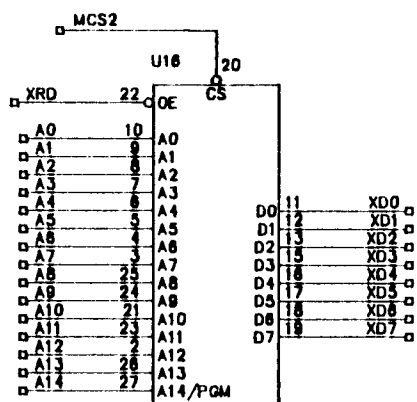
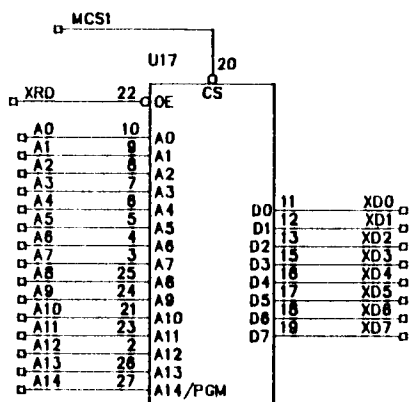
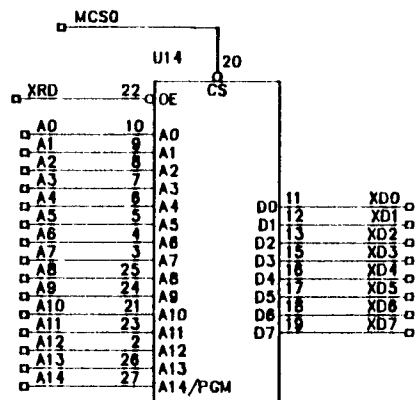
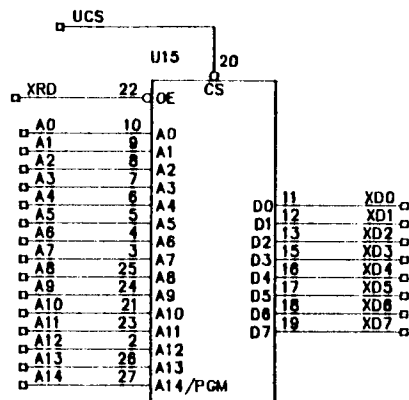


TABLE 1

DEVICE TABLE

REFERENCE DESIGNATION	DEVICE TYPE	POWER PINS			
		GND	+5V	+12V	-12V
U1	74LS373	10	20		
U2	74LS373	10	20		
U3	74LS37	7	14		
U4	74LS373	10	20		
U5	74LS245	10	20		
U6	74LS38	7	14		
U7	60188-6	28, 60	9, 43		
U8	8288	10	20		
U9	74LS37	7	14		
U10	74LS30	7	14		
U11	74LS174	8	16		
U13	74LS74	7	14		
U14	2784/128	14	28		
U15	2784/128	14	28		
U16	2784/128	14	28		
U17	2784/128	14	28		
U18	8251A	4	28		
U19	MC1488	7		14	1
U20	MC1488	7		14	1
U21	8251A	4	28		
U22	MC1489A	7	14		
U23	MC1489A	7	14		

Notes

1. Dimensions in inches.

2. Tolerances:

.XX ±

.XX ±

Angles ±

Signature

Designed

Drawn

APPR.

APPR.

Date

Title

80188
CPU CARD

Revision

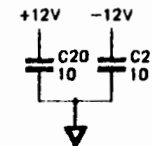
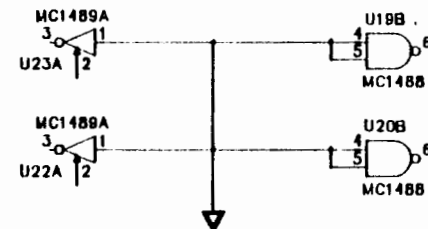
1

Document Number

Scale

—

Sheet 2 of 3



Notes 1. Dimensions in inches. 2. Tolerances: .XX ±— .XX ±— Angles ±—	Signature	Date	Title <div style="text-align: center; font-size: 1.5em;">80188 CPU CARD</div>	
	Designed			
	Drawn			
	APPR.		Revision	Document Number
	APPR.		1	
		Scale	—	Sheet 3 of 3

APPENDIX C

CIRCUIT DIAGRAMS : TAPE INTERFACE MODULE

The module is composed of two SABUS cards. These cards are connected via a 40 way ribbon cable of which two lines are also connected to the CPU card. These are the DMA request lines. Connection to SABUS is made only via the SABUS connector on card 1, with the connector on card 2 being used only as a mechanical connection. This allows both cards to be simultaneously debugged with only one extender card. Card 1 also has the write data and control connectors to the tape transports, while card 2 contains the read data connector. Note that in the description which follows all components on card 1 have a number beginning with 1 (eg. U101) and all components on card 2 have a number beginning with 2 (eg. U201).

1. SABUS interface. The SABUS interface's function is to provide address decoding, as well as buffering for the data bus and control lines. U102, U103 and U104 decode the base address of the module, while U107 and U118b provide enable lines to each addressable device in the module. The base address of the module is wire-wrap selectable. U101 is the

data bus buffer. All of the components related to the SABUS interface are located on card 1.

2. Output buffers. The output buffer section of the module provides both buffering and latching of signal to be outputted to the tape transports. All of these devices reside on card 1. This section may be further subdivided into three subsections :

2.1. Data output latches. These latches, composed of U113 and U119, latch the output data to the tape drives. The eight data bits are latched by U119, and the parity bit as well as the two strobe enable bits (as described earlier) are latched by U113. The data and parity bits are buffered and transmitted to the tape transport, while the strobe enable bits are gated with signals from the timing generator to provide write and write-reset strobes to the transports. Data transfer to these latches is by DMA, as described earlier.

2.2. Configuration latch. This latch, U117, sets the configuration of the module. It has the following outputs :

2.2.1. Drive select lines. Two bits are used to select the tape transport. These lines are decoded by U118a, a two to four line decoder. The resulting one of four select lines are buffered and used to select the appropriate tape transport.

2.2.2. Count select lines. These two lines are fed to the timing section of the module, and will be described in a later section.

2.2.3. Forward/reverse select line. This line selects the direction of tape motion.

2.2.4. Read/write select line. This line selects whether a read or a write operation is to be performed.

2.3. Command output latch. It is the function of the command output latch to provide signals to the module which may be used to drive edge sensitive inputs. In order to achieve this a bit addressable latch is used, so as to allow only one bit at a time to be updated without raising the possibility of unwanted transitions on any other bit. This latch, U108, provides four edge sensitive signals (SWS, RWC, OFFC and RTH) direct to the tape drive as well as a GO (start operation) and a board reset signal to the rest of the module. The output of the latch are reset to a logical zero by the SABUS reset signal, which sets the board reset active, and all other signals inactive. This prevents the module from generating any unwanted signal during power up.

3. Input buffers. The input buffer and latch section resides on card 2, and consists of two subsections :

3.1. Data input buffers and latches. U203 latches data from the tape transports, while U204a latches the parity bit. U201 buffers this bit onto the data bus, in

addition to several status bits. These status bits include state information from the format recognition section, and a bit from the DMA section which indicates the occurrence of a DMA error. It should be noted that these status bits are not latched by the RDS strobe, as the state of the format recognition module after a character is read in, rather than when it is read in, is required for check character recognition. This status information is stored for each character which is read in. Transfer from both these buffers is always by DMA, as is that of the output data latches.

3.2. Status input buffer. This buffer, U202, buffers several lines which are taken directly from the tape transports, as well as two signals from the timing module itself. These signal are a run signal, which indicates whether the module is executing a command, and an EOF signal, which indicates that a tape mark was passed during the process of the last command.

4. Format recognition section. The function of this section, which resides entirely on card 2, was described above. It consists essentially of a state machine, timing circuits and state decoding logic. The state machine consists of three JK flip-flops (U211b, U217a and U217b) and their associated gates. The state machine is clocked by the RDS strobe from the tape transports, and has two inputs which indicate timing information. These two input are SG (short gap) and LG (long gap). SG is active when a strobe has not been detected for two character clocks, and LG when a strobe has not been detected for ten character clocks. Both of these signal are generated by programmable timers (counters 1 and 2 of U110), so allowing operation with tape transport of various speeds. U218c and U219c decode a signal denoting

that the tape head is over an inter-record gap, while U218a and U219a detect the passing of a tape mark.

5. Timing generation section. This section generates signals which require precision timing, such as the tape motion signals and the write strobes. The timing section consists of several sections :

5.1. Master clock. The master clock consists of Q1 and its associated components, which form a highly stable 10 MHz oscillator. The circuit is a Colpitts oscillator, which was chosen above logic gate oscillators for its superior accuracy and start-up performance [13]. U112A divides this signal down to a 5 MHz square wave, which is used to derive all other module timing.

5.2. Character clock generation. The character clock is generated by dividing the master clock by a programmable ratio. This division is done by counter 0 of U110, and the pulse width of the resulting signal is set to precisely 2 uS by U111. This signal is used to generate write strobes, as well as for timing purposes.

5.3. Start counter. This counter (counter 0 of U105) sets the delay from the start of an operation to the beginning of the count phase of the operation. This only has real significance in the case of write operations, when this is when the first character is written. For all other operations this counter, although still in the timing chain, is set to a small nominal value. The start of this timing period also sets the run flip-flop, U112b. When this flip-flop is set, tape motion is enabled, setting either the

synchronous forward command line or the synchronous reverse command line active, dependant on the setting of the forward/reverse status line described above. The output of this flip-flop is also available for reading by the CPU, to ascertain whether or not the operation is complete. Note that this bit goes inactive when the tape motion commands go inactive, but that the tape motion has not yet ceased at this point.

5.4. Main counter. This counter, counter 1 of U105, counts either write strobes, read strobes or blocks, as defined by the count select line. The time during which this counter is active is known as the count phase, and this phase may be ended either by the counter reaching zero or by a programmable end condition. This end condition is programmed by the same count select lines as the items to be counted. Both the selection of the count and the selection of an end condition are done by U220, which is a dual four way multiplexer. Three combinations are valid for the count select lines. These are as follows :

5.4.1. Count select of 00. This value selects the counting of write strobes, and no end condition. This is used for write operations, which always end only when all characters have been written.

5.4.2. Count select of 01. This value selects the counting of read strobes, and an end condition of encountering a inter-record gap. This is used for read block operations, when the operation is normally terminated at the end of a block (e.i. when an inter-record gap is encountered) or when some maximum number of characters is exceeded.

This last condition is an error, and the operation is ended to avoid buffer overflow.

5.4.3. Count select of 10. This value selects the counting of inter-record gaps encountered, and an end condition of passing a tape mark. This is used for space operations, where a specified number of blocks must be spaced, and the operation must be aborted on encountering a tape mark.

5.5. Stop counter. Once the end of an operation is encountered, a delay must occur before the tape motion command is deactivated, in order to ensure that the tape head stops at the correct point relative to the end of the last block or tape mark. This is accomplished by counter 2 of U105, which resets the run flip-flop on completion of its count, thus taking the motion command lines inactive and informing the CPU that the operation is complete.

6. DMA request generation section. This section generates two DMA request signals which are passed to the CPU card. U210b and U207b are set to request DMA cycles for the data output latch and the parity and strobe information latch respectively. Both of these flip-flops are set by either a write strobe (including dummy writes) or by the GO signal. The GO signal loads the data for the the first write into the latches. The flip-flops are reset by a write to their respective latches. U211a and U210a perform a similar function for the read data, but in this case the flip-flops are set only by the read strobe. U204b and U207a detect DMA overrun errors which occur in the event of a DMA cycle being required before the previous cycle is complete.

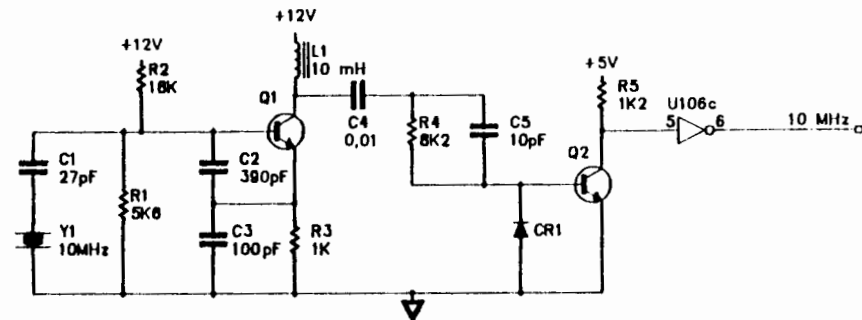
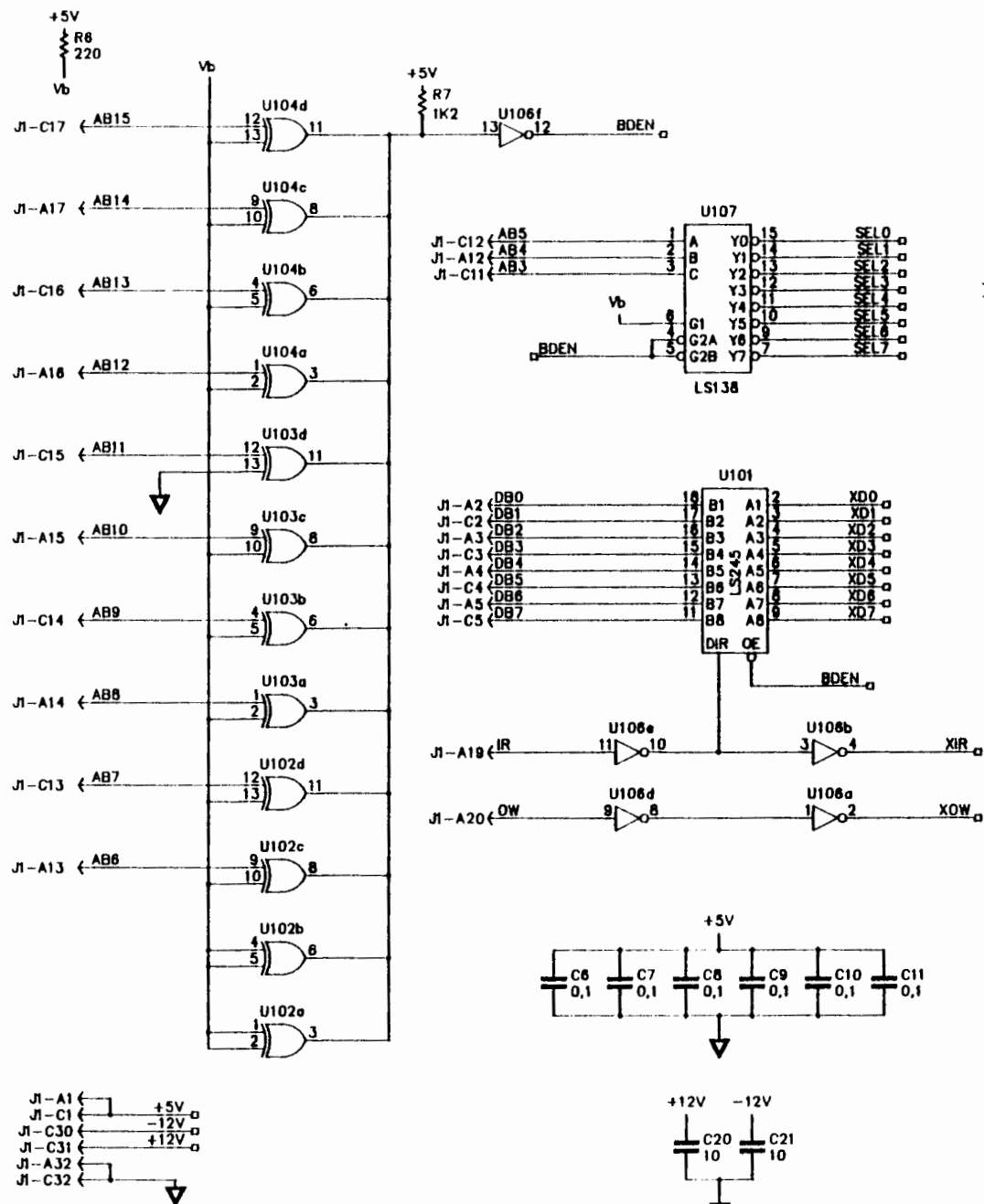
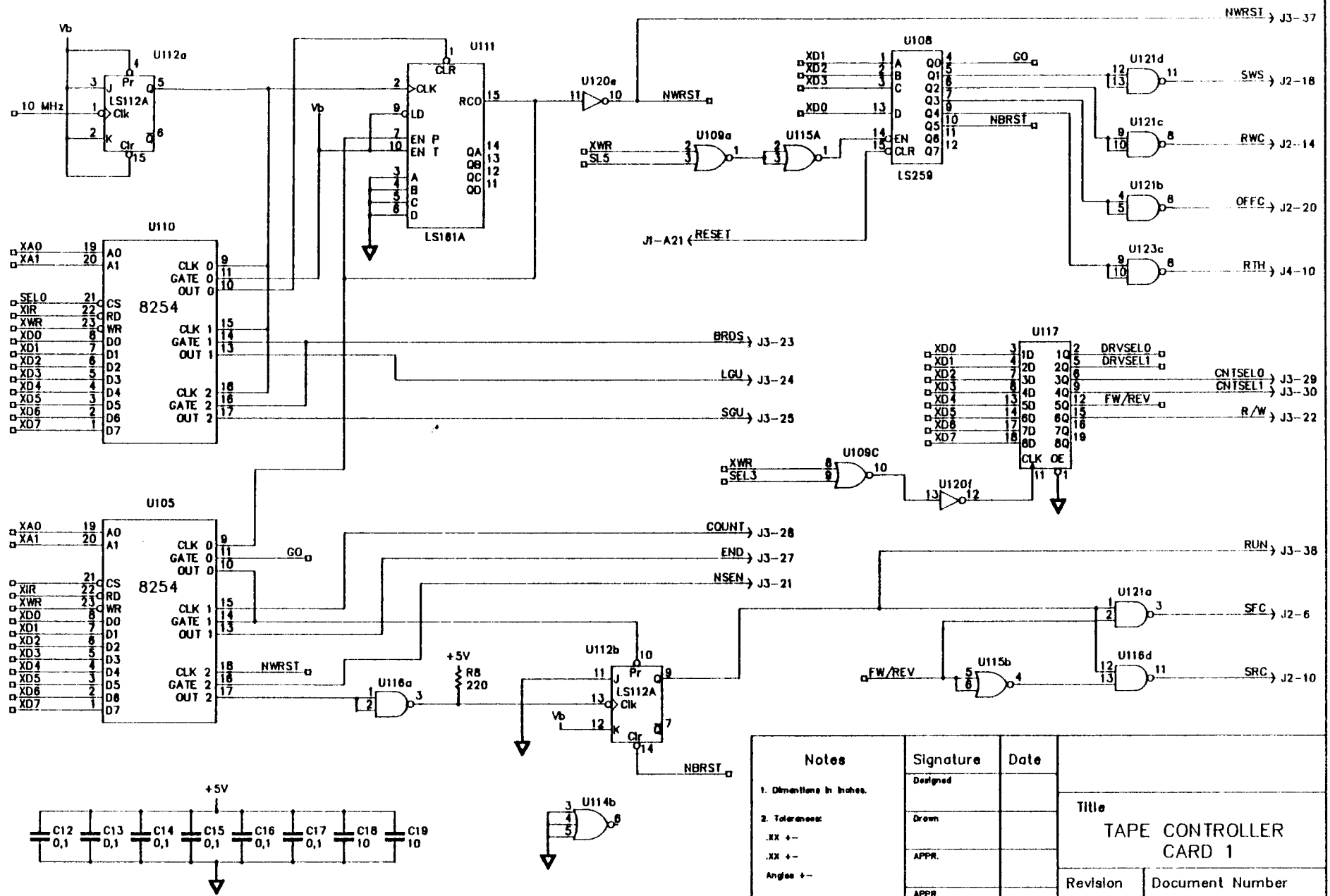


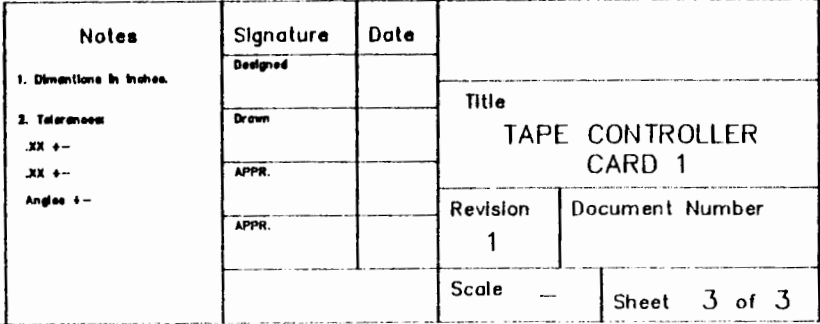
TABLE 1

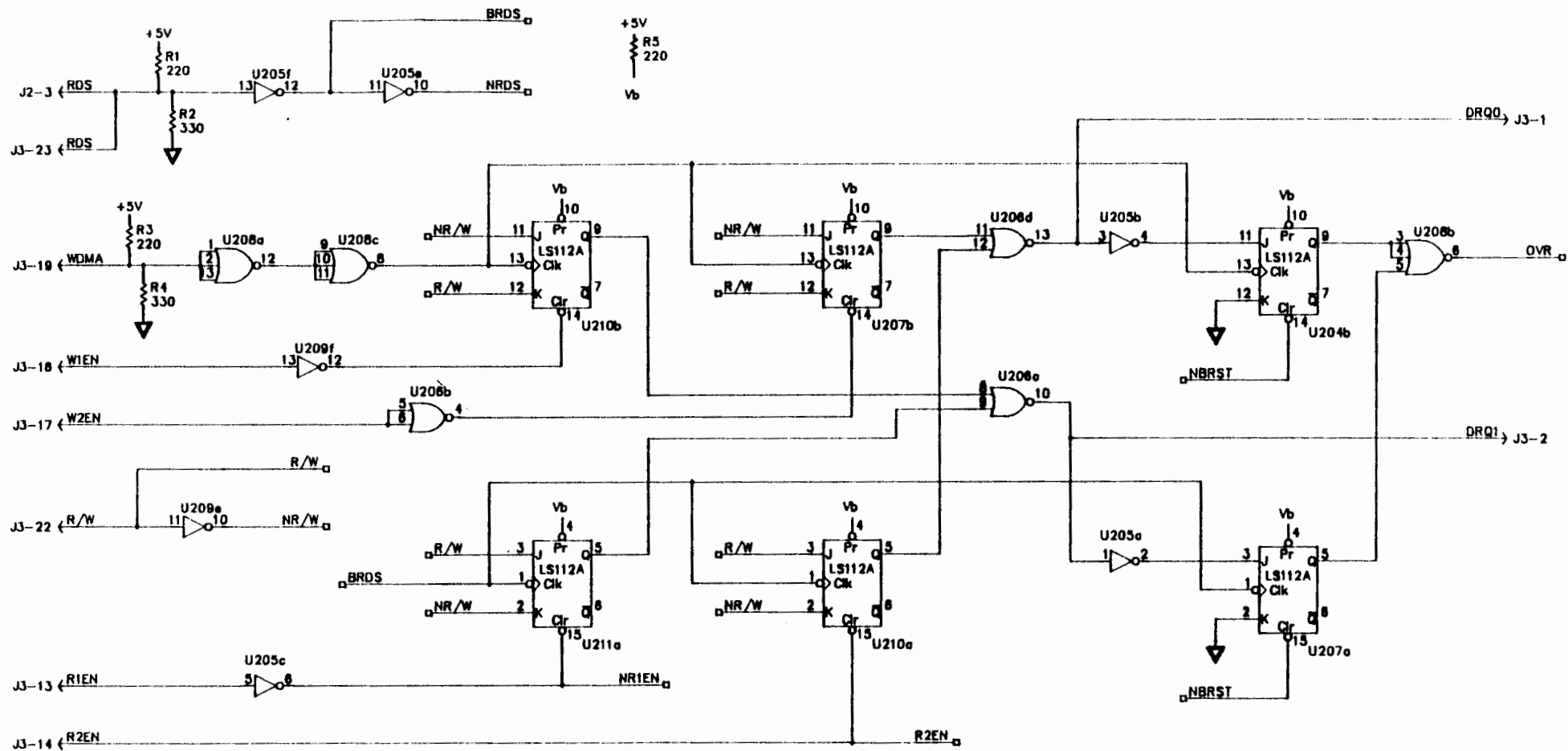
REFERENCE DESIGNATION	DEVICE TYPE	POWER PINS			
		GND	+5V	+12V	-12V
U101	74LS245	10	20		
U102	74LS138	7	14		
U103	74LS138	7	14		
U104	74LS138	7	14		
U105	8254	12	24		
U106	74LS04	7	14		
U107	74LS138	8	16		
U108	74LS259	8	16		
U109	74LS02	7	14		
U110	8254	12	24		
U111	74LS161A	8	16		
U112	74LS112A	8	16		
U113	74LS373	10	20		
U114	74LS27	7	14		
U115	74LS02	7	14		
U116	74LS36	7	14		
U117	74LS374	10	20		
U118	74LS139	8	16		
U119	74LS373	10	20		
U120	74LS04	7	14		
U121	74LS36	7	14		
U122	74LS36	7	14		
U123	74LS36	7	14		
U124	74LS36	7	14		
U125	74LS36	7	14		

Notes 1. Dimensions in inches. 2. Tolerances: .XX ± .XX ± Angles ±	Signature	Date	Title TAPE CONTROLLER CARD 1	
	Designed			
	Drawn		Revision	Document Number
	APPR.		1	
APPR.		Scale	Sheet 1 of 3	

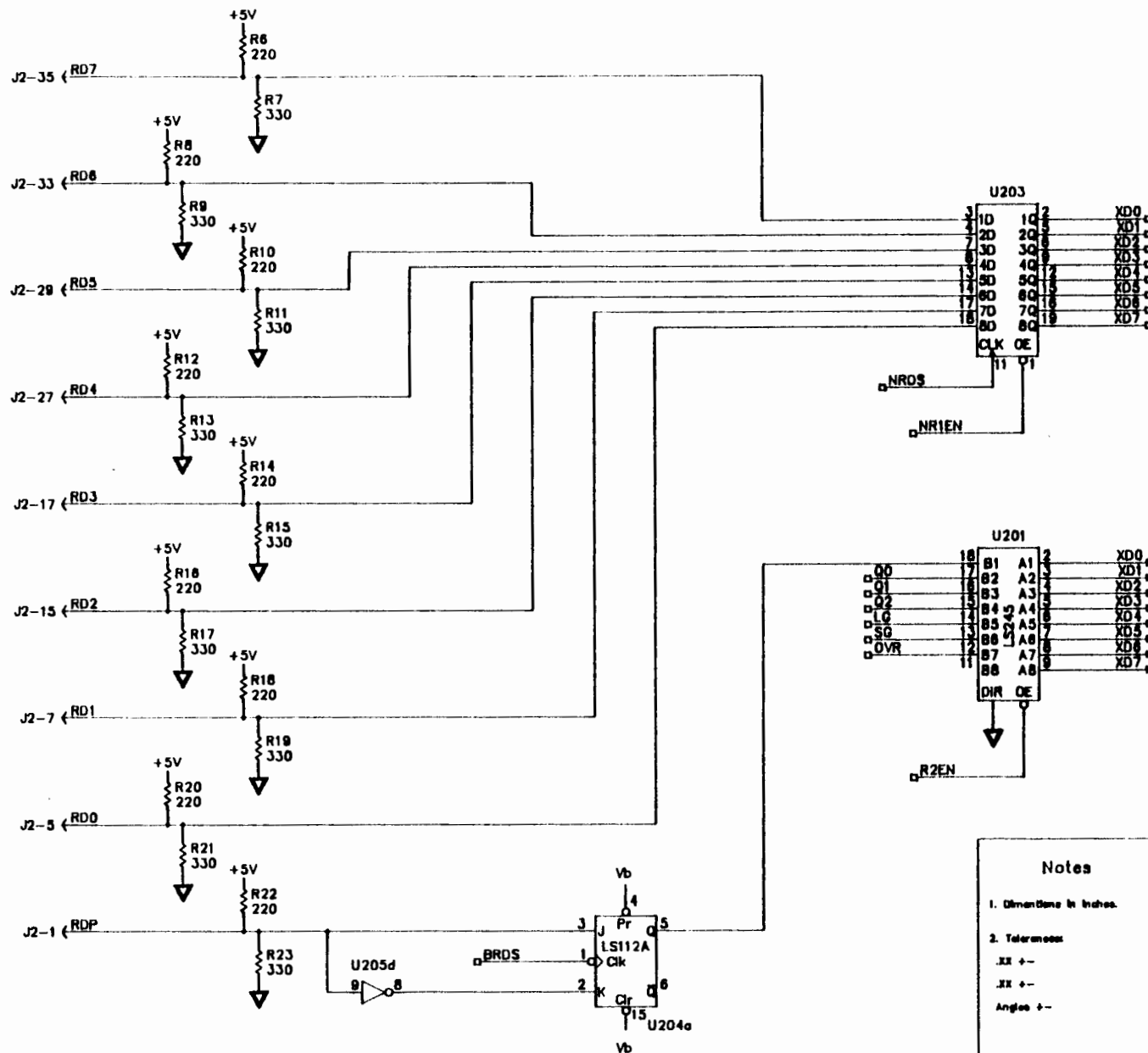


Notes		Signature	Date	Title	
1. Dimensions in inches.		Designed		TAPE CONTROLLER CARD 1	
2. Tolerances:		Drawn			
.XX ±		APPR.			
Angles ±		APPR.			
				Revision	Document Number
				1	
				Scale	Sheet 2 of 3

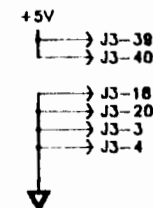




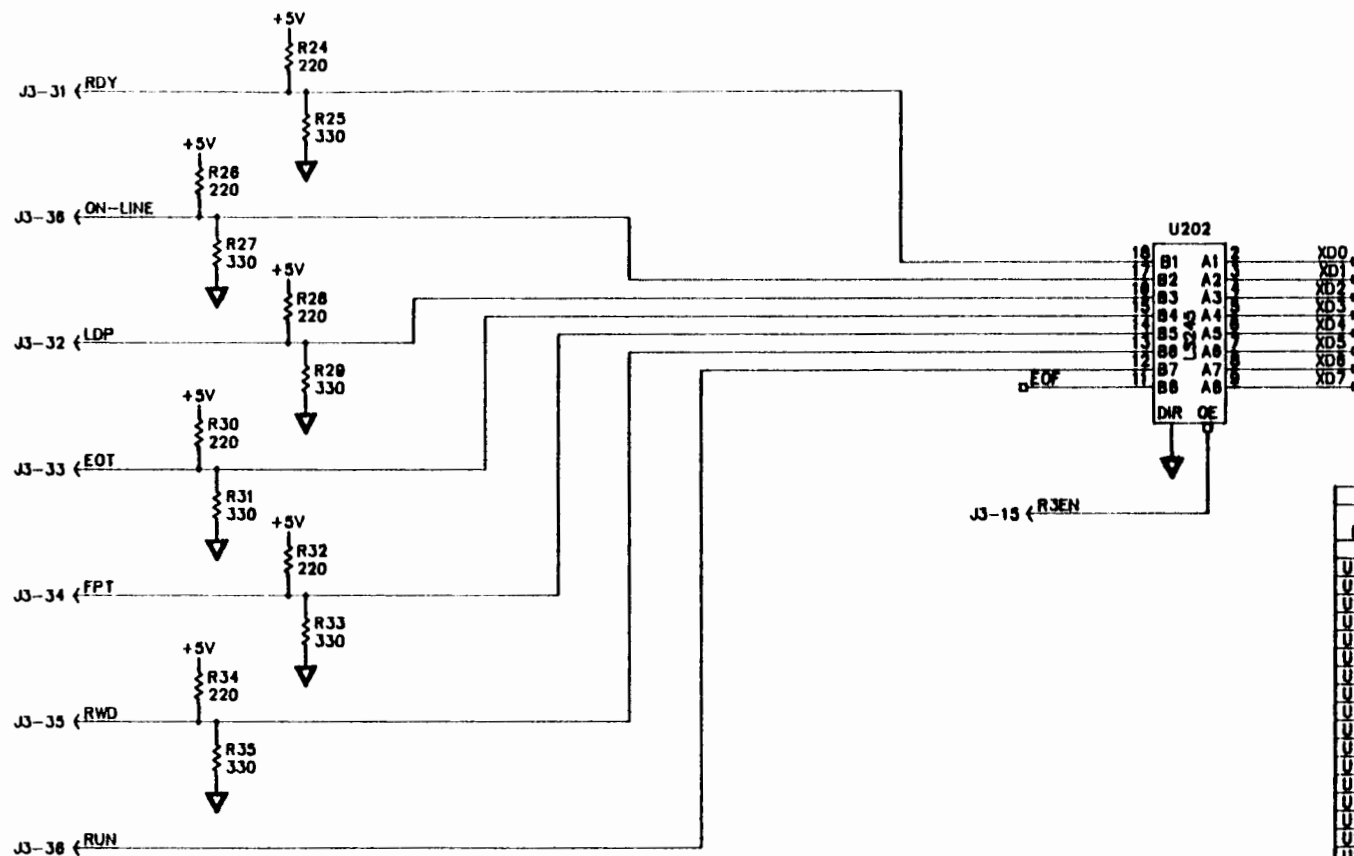
Notes	Signature	Date	Title	
	Designed			
	Drawn		TAPE CONTROLLER CARD 2	
	APPR.		Revision	Document Number
	APPR.		1	
			Scale	Sheet 1 of 4



XD0	XD0	J3-5
XD1	XD1	J3-6
XD2	XD2	J3-7
XD3	XD3	J3-8
XD4	XD4	J3-8
XD5	XD5	J3-9
XD6	XD6	J3-10
XD7	XD7	J3-11
XD7	XD7	J3-12



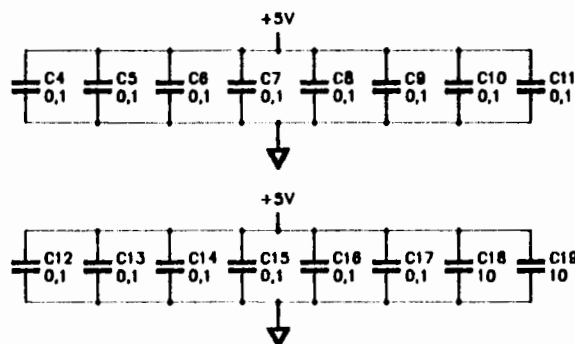
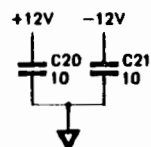
Notes	Signature	Date	Title	
	Designed			
	Drawn		Revision	
	APPR.			
			1	Document Number
			Scale —	Sheet 3 of 4



J3-15 ← R3EN

TABLE 1
DEVICE TABLE

REFERENCE DESIGNATION	DEVICE TYPE	POWER PINS			
		GND	+5V	+12V	-12V
U201	74LS245	10	20		
U202	74LS245	10	20		
U203	74LS374	10	20		
U204	74LS112A	8	16		
U205	74LS04	7	14		
U206	74LS02	7	14		
U207	74LS112A	8	16		
U208	74LS27	7	14		
U209	74LS04	7	14		
U210	74LS112A	8	16		
U211	74LS112A	8	16		
U212	74LS10	7	14		
U213	74LS10	7	14		
U214	74LS10	7	14		
U215	74LS00	7	14		
U216	74LS10	7	14		
U217	74LS112A	8	16		
U218	74LS10	7	14		
U219	74LS00	7	14		
U220	74LS153	8	16		
U221	74LS00	7	14		



Notes

1. Dimensions in inches.

2. Tolerances:

.XX ±

.XX ±

Angles ±

Signature	Date	Title TAPE CONTROLLER CARD 2	
Designed			
Drawn		Revision 1	Document Number
APPR.			
APPR.		Scale	Sheet 4 of 4

APPENDIX D

DATA LINK FORMAT INFORMATION

Communication on the RS-232 data link between the host system and the tape controller takes place in frames and blocks. The host transmits a command block to the controller, to which the controller replies with a response block. To ensure the error free transmission of these blocks a frame protocol is used. This appendix describes the formats of the blocks and frames used.

1. Data protocol

The link makes use of a character orientated frame protocol of the PAR (Positive Acknowledgement with Retransmission) type. The protocol is designed to allow the error-free transmission of an arbitrary number of 8-bit bytes.

2. Frame format

2.1. Character format. Each character is transmitted in conventional 8-bit/no parity serial format. Baud rate is not specified, but is normally 9600.

2.2. Special characters. Certain characters have special meaning in frames. Their use will be described later. These characters are the following (numeric values in octal).

DLE :	20
ACK :	6
NAK :	25
START :	2
CONT :	3
END :	4

2.3. Frame format. All frames have a common format :

DLE	The first character in a frame is always a DLE character. This character, together with the lead character, allows easy recognition of the start of a frame.
Lead character	The lead character identifies the type of frame. Possible values are described below.
Frame number	The Frame number is always either 0 or 1. In the case of data frames this alternates sequentially, and

in the case of Ack and Nak frames this refers to the frame being acknowledged.

Data	A variable number of data characters, up to a maximum of 64, may be transmitted. Character stuffing of DLE characters is performed.
DLE	This single DLE character indicates the end of the data section of the frame, and is always present.
Terminating character	The terminating character serves to distinguish the frame end from the frame beginning, as well as to indicate whether or not this frame is the last in the block. The valid terminating characters are described below.
CRC low byte	In order to provide error detection for the frame a CRC is appended. The CRC is CCITT-16 standard. This CRC is performed on all characters in the frame, and starts from an initial value of FFFF hex. The CRC is transmitted as two characters, the least significant byte being transmitted first.
CRC high byte	This is the second part of the CRC described above.

2.3.1. Three types of frames exist, and the frame type is indicated by the lead character. These are the following :

2.3.1.1. Data frames. Data frames are indicated by a START character in the lead position. Data frames carry a variable number of data characters, and are terminated by either a CONT or an END character. If terminated by a CONT character then the next frame is a continuation of the current block, and if terminated by an END character then the frame is the last frame in the current block.

2.3.1.2. Ack frames. Ack frames are indicated by an ACK character in the lead position, and are transmitted to indicate the the frame whose number is in the frame number field was received correctly. Ack frames never contain data, and are terminated by an END character.

2.3.1.3. Nak frames. Nak frames are indicated by an NAK character in the lead position, and are transmitted to indicate the the frame whose number is in the frame number field contained an error. Nak frames never contain data, and are terminated by an END character.

2.4. State values. The use of state driven software for both the transmission and reception of frames was discussed in the main text. The following states are used in both the PDP-11 and tape controller software :

2.4.1. Transmit states. These states are defined by the character which will be transmitted as soon as the transmit data port becomes empty.

2.4.1.1. State 0. Start condition. A DLE is to be transmitted. Next state is state 1.

2.4.1.2. State 1. Transmit the lead character. Next state is state 2.

2.4.1.3. State 2. Transmit the frame number. Next state is state 3 if there is data to transmit, else state 5.

2.4.1.4. State 3. Transmit a data character. Next state is state 4 if the character was a DLE, else state 5 if the character was the last data character in the frame, or else the state remains at 3.

2.4.1.5. State 4. Transmit a DLE. Next state is state 5 if the no data is left to transmit, or else state 3.

2.4.1.6. State 5. A DLE is to be transmitted. Next state is state 6.

2.4.1.7. State 6. The terminating character is to be transmitted. Next state is state 7.

2.4.1.8. State 7. The low byte of the CRC is to be transmitted. Next state is state 8.

2.4.1.9. State 8. The high byte of the CRC is to be transmitted. Next state is state 0, ready for the next frame.

2.4.2. Receive states. These states are defined by the character which the device expects to receive next.

2.4.2.1. State 0. Start condition. A DLE is to be received. Next state is state 1 but only if the character was a DLE, else the state remains at 0.

2.4.2.2. State 1. Receive the lead character. Next state is state 2, if the character was a valid lead character. If the character was not valid, then the state reverts to 0.

2.4.2.3. State 2. Receive the frame number. Next state is state 3.

2.4.2.4. State 3. Receive a data character. Next state is state 4 if the character is a DLE, or else the state remains at 3.

2.4.2.5. State 4. Receive either a DLE or a terminating character. Next state is state 3 if the character was a DLE, or else state 5.

2.4.2.6. State 5. The low byte of the CRC is to be received. Next state is state 6.

2.4.2.7. State 6. The high byte of the CRC is to be received. Next state is state 0, ready for the next frame.

3. Block format

Two block formats exist; one for command blocks and one for response blocks.

3.1. Command blocks. Command blocks contain five fields. These are the following :

Opcode	This field is one byte long, and contains the operation to be executed, as well as certain other information. A detailed description of this field is given below.
Unit number	This field, which is one byte long, contains the logical unit number to which the command block refers.
Status in	This is a 16 bit field which contains status information. The definition of each bit is given below.
Count in	This is a 16 bit field which normally contains a count value, but contains a status value for certain operations. This is discussed below.
Data	This field, which is from 0 to 8192 bytes long, contains data for write operations. For all other operations it is empty.

3.1.1. Opcode. This field is one byte long, and contains the following information.

3.1.1.1. Bits 0-5. The command to be executed is encoded into the lower 6 bits of the opcode. Ten command are supported. These are the following (numeric codes in decimal) :

3.1.1.1.1. RLB (1). Read logical block. The next block is read in, and transmitted to the host. The count field contain the maximum number of characters to be read in. If a greater number is read in then an error occurs and no data is returned to the host.

3.1.1.1.2. WLB (2). Write logical block. The transmitted block of data is written to tape. The number of data bytes is contained in the count field.

3.1.1.1.3. EOF (3). write end-of-file mark. A tape mark is written. The count field is not used.

3.1.1.1.4. RWD (4). Rewind. The tape on the unit in question is rewound. The count field is not used.

3.1.1.1.5. RWU (5). Rewind and off-line. The unit is rewound and set off-line. The count field is not used.

3.1.1.1.6. SPB (6). Space blocks. The tape is spaced by the selected number of blocks. The count field contains the two-compliment value of the number of blocks to space. The space operation is terminated on encountering a tape mark.

3.1.1.1.7. SPF (7). Space files. The tape is spaced by the selected number of files. The count field contains the two-compliment value of the number of files to space. ANSI formatted tapes

files are delimited by end-of-file records, but unformatted tapes are delimited by tape marks. The end of data on a tape is known as the logical end of volume. This is denoted by a double tape mark for unformatted tapes, and an EOVS record followed by a double tape mark for ANSI tapes. Note that the end of volume condition is not detected for ANSI tapes. Space file operations are terminated on encountering the logical end of volume condition in the case of unformatted tapes.

3.1.1.1.8. STC (8). Set tape characteristics. This is used in order to set bits in the status word. See below for a description of which bits are affected. The count field contains the bit pattern to be inserted into the status word. Note that this function must be carried out by the tape controller because although the status word is stored in the host, the controller must check that the host does not attempt to operate the tape units in unsupported modes (eg at the wrong bit density).

3.1.1.1.9. SEC (9). Sense tape characteristics. This reads the status word. See below for a discussion of the return value. The count field is not used.

3.1.1.1.10 SMO (10). Mount and set tape characteristics. This is identical to STC, except that if the unit is not ready and at the BOT tab, then an error results. The count field contains the bit pattern to be inserted into the status word.

3.1.1.2. Bit 6. Bit 6 is set if the mounted tape is ANSI formatted, and cleared otherwise. This information is necessary to determine file delimiters, as discussed above.

3.1.1.3. Bit 7. Bit 7 is set if retries for operations which failed are disallowed. If this bit is not set then the tape controller will normally retry a failed operation up to 10 times.

3.1.2. Status field. This is a 16 bit word, which contains status information relation both to the characteristics of the tape transports, and the current status of the transport. The characteristics bits can be set by the STC and SMO commands described above. A description of each bit will now be given. Note that the description provided below only describes bits which are relevant to the tape controller. The PDP-11 assigns meaning to all of these bits, but only some are used by the tape controller. Only bits 6 and 7 can be set by STC and SMO, and have meaning to the controller. Note however that bits 0,1,3 and 11 can also be set and reset, but are ignored by the controller. This is done for reasons of compatibility with the TM-11.

<u>Bit</u>	<u>Meaning when set</u>
0	Not used.
1	Not used.
2	Not used.
3	Not used.
4	Tape is past the EOT tab.

5	Last tape command encountered an EOF (tape mark).
6	Writing is prohibited.
7	Writing with an extended inter-record gap is prohibited. This means that no error recovery will be attempted on write operations.
8	Select error.
9	Unit is rewinding.
10	Tape is physically write-locked.
11	Not used.
12	Not used.
13	Tape is at BOT.
14	Tape is at the end of volume. Note that this condition is detected only for unformatted tapes, and that the head is positioned between the tape marks which denote it.
15	Not used.

3.2. Response blocks. Response blocks consist of four fields. These are the following :

Status	This is a 16 bit field, and contains the same information as the incoming status in the command field, as described above.
--------	--

Count	This normally contains an integer field, but occasionally contains status information. This is further discussed below.
Return code	This is a 16 bit word which contains a numeric code. This indicates the result of the operation. Possible values are given below.
Data	This field varies from 0 to 8192 bytes. It only exists in response to read commands. Note that if the transport encounters a EOF mark then two bytes are returned, each of which are the numeric value of the character used in a tape mark. This is done for TM-11 compatibility purposes.

3.2.1. Count value. The meaning of the value in the count field is described for the various operations. Note that operations not listed here do not make use of this field. The meaning of the mnemonics are described above. Numeric values are in two's compliment form.

RLB	Number of bytes transferred.
WLB	Number of bytes transferred.
SPB	Number of blocks spaced over.
SPF	Number of files spaced over.
SEC	Status word as described above.

3.2.2. Return code. The various possible return codes are described below. All numeric values are in octal.

IS.SUC (1)	The command was successfully completed.
IE.EOT (302)	The tape is past the EOT tab. Note that this means that the operation was successfully completed.
IE.FHE (305)	The unit was not ready, or a catastrophic hardware error occurred, so preventing the operation from being attempted.
IE.BBE (310)	A bad block was encountered, and the error was not recoverable.
IE.DAO (363)	Data overrun. A block which was read in was larger than the the stated size.
IE.WLK (364)	A write operation was attempted on a unit which was either physically write-locked, or had the write-lock bit in the status word set.
IE.EOF (366)	An EOF (tape mark) was encountered.
IE.SPC (372)	Illegal buffer size. A byte count of less than 8 was specified for a read operation, or a byte count of less than 14 for a write operation.
IE.VER (374)	Irrecoverable parity or CRC error.
IE.IFC (376)	Illegal function request.

APPENDIX E

SOFTWARE LISTINGS : PDP-11 DEVICE DRIVER

<u>Module</u>	<u>PAGE</u>
Device driver	E-2
Device tables	E-19

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3

```

1      .TITLE MADRV
2      .IDENT /02.05/
3
4
5      ;
6      ; VERSION 2.2
7      ;
8      ; A.D. MC GUFFOG      MARCH 1985
9      ;
10     ; PREVIOUSLY MODIFIED BY:
11     ;
12     ; MODIFIED BY:
13     ;
14     ;
15     ;
16     ; UCT SERIAL 9-TRACK TAPE UNIT DRIVER
17     ;
18     ; THIS DRIVER INTERFACES WITH THE TAPE TRANSPORT VIA A DL11 AT 9600 BAUD.
19     ; COMMANDS, DATA AND DEVICE STATUS ARE FORMATTED INTO STRUCTURED
20     ; PACKETS OF BYTES IN ACCORDANCE WITH A SERIAL PROTOCOL.
21     ;
22     ; MACRO LIBRARY CALLS
23     ;
24     .MCALL HWDDF$,PKTDF$,UCBDF$,SCBDF$
25 000000 HWDDF$      ;DEFINE HARDWARE REGISTERS
26 000000 PKTDF$      ;DEFINE I/O PACKET OFFSETS
27 000000 UCBDF$
28 000000 SCBDF$
29
30     ;
31     ; EQUATED SYMBOLS
32     ;
33 000000 LD$NA == 0 ;LOADABLE DRIVER
34
35     ;
36 000200 TRDY= 200 ;TRANSMITTER READY
37 000100 TIE= 100 ;TRANSMITTER INTERRUPT ENABLE
38 000100 RIE = 100
39
40     ;
41 000006 ACK = 6
42 000020 DLE = 20
43 000025 NAK = 25
44 000002 START = 2
45 000003 CONT = 3
46 000004 END = 4
47 000027 RESET = 27
48
49     ;
50 000000 RCS = 0
51 000004 TCS = 4
52 000002 RBUF = 2
53 000006 TBUF = 6
54
55     ;
56 177777 CRCINIT = 177777
57 170270 CRCFIN = 170270

```

MACRV MACRO M1200 20-JUN-86 15:43 PAGE 3-1

```

58      ;
59      ;
60      000005      RETRYS =      5
61      ;
62      ;
63      000001      TR.STR =      1
64      000002      TR.DATA =      2
65      000004      TR.END =      4
66      000010      TR.ERR =     10
67      000020      TR.OP =     20
68      000040      TR.BSY =     40
69      000100      TR.ACR =    100
70      000200      TR.DIN =    200
71      ;
72      ;
73      000003      RESETFNUM      =      3
74      ;
75      000200      NO.RETRY      =     200
76      ;
77      000100      LAB.TAPE      =     100
78      ;
79      ;
80      000200      BUFLN      =     128.
81      000100      NUMDAT      =     64.
82      ;
83      ;
84      000001      RLBCMD      =      1.
85      000002      WLBCMD      =      2.
86      000003      EDFCMD      =      3.
87      000004      RWCMD      =      4.
88      000005      RWUCMD      =      5.
89      000006      SPBCMD      =      6.
90      000007      SPFCMD      =      7.
91      000010      STCCMD      =      8.
92      000011      SECCMD      =      9.
93      000012      SMCMD      =     10.
94      ;
95
96 000000 000000      CNTBL:      .WORD  0
97 000002 000000      TLEN:      .WORD  0
98 000004 000000      COUNT:     .WORD  0
99 000006 000000      TINDEX:    .WORD  0
100 000010 000000     RINDEX:    .WORD  0
101 000012 000000     RETCOUNT: .WORD  0
102 000014 000000     ERRCODE:   .WORD  0
103 000016 000000     RETRYCNT:  .WORD  0
104 000020 000000     RSTATE:    .WORD  0
105 000022 000000     TSTATE:    .WORD  0
106 000024 000000     RCRC:      .WORD  0
107 000026 000000     TCRC:      .WORD  0
108 000030 000000     TCRCV:     .WORD  0
109 000032 000      OPCODE:      .BYTE  0
110 000033 000      TRSTATUS:    .BYTE  0
111 000034 000      FNUMIN:      .BYTE  0
112 000035 000      FNUMOUT:     .BYTE  0
113 000036 000      RFCHAR:      .BYTE  0
114 000037 000      RTCHAR:      .BYTE  0

```

MAORV MACRO M1200 20-JUN-86 15:43 PAGE 3-2

```

115 000040      000      TFCHAR:      .BYTE  0
116 000041      000      TTCHAR:      .BYTE  0
117 000042 000000      TFNUM:      .WORD  0
118 000044 000000      RFNUM:      .WORD  0
119      ;
120      ;
121      ;NOW THE FRAME BUFFER SPACE
122      ;
123      ;
124 000046      TDATA:      .BLKB  BUFLN
125      ;
126 000246      RDATA:      .BLKB  BUFLN
127      ;
128      ;
129      ;NOW THE STATE TABLES
130      ;
131      ;
132 000446 002236'      RSTTBL:      .WORD  RCFXDL
133 000450 002276'      .WORD  RCFCHAR
134 000452 002402'      .WORD  RCFNUM
135 000454 002436'      .WORD  RCDATA
136 000456 002502'      .WORD  RCDLE
137 000460 002424'      .WORD  REXIT
138 000462 002542'      .WORD  RCCRC
139      ;
140      ;
141 000464 001472'      TSTTBL:      .WORD  TXOLE
142 000466 001500'      .WORD  TXFCHAR
143 000470 001524'      .WORD  TXFNUM
144 000472 001546'      .WORD  TXDATA
145 000474 001722'      .WORD  TXDLE
146 000476 001472'      .WORD  TXOLE
147 000500 001744'      .WORD  TXTCHAR
148 000502 001752'      .WORD  TXCRCLO
149 000504 001774'      .WORD  TXCRC
150      ;
151      ;
152      ;
153      .MACRO  TBLENT  COM,CMD,TYPE
154
155          .WORD  COM
156          .BYTE  CMD, TYPE
157
158      .ENDM  TBLENT
159      ;
160      ; LOCAL DATA STORAGE
161      ;
162      ;
163      ;
164 000506      LGFCN:  TBLENT  IO.RLB, RLBCMD, 0
165 000512      TBLENT  IO.WLB, WLBCMD, 0
166 000516      TBLENT  IO.EOF, EOF CMD, 0
167 000522      TBLENT  IO.RWD, RWDCMD, 1
168 000526      TBLENT  IO.RWU, RWUCMD, 1
169 000532      TBLENT  IO.SPB, SPBCMD, 0
170 000536      TBLENT  IO.SPF, SPFCMD, 0
171 000542      TBLENT  IO.STC, STCCMD, -1

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-3

```

172 000546          TBLENT IO.SEC, SECCMD, -1
173 000552          TBLENT IO.SMO, SMOCMD, -1
174          000556' ENOLGF =
175          ;
176          ;
177 000556 000566' $MATBL:: .WORD MAINI
178 000560 003224'          .WORD MACAN
179 000562 003244'          .WORD MAOUT
180 000564 003242'          .WORD MAPWF
181          ;
182          ;
183          LOCAL BIT ASSIGNMENTS
184          ;
185          ;
186          **-MAINI- TAPE INITIATOR
187          ;
188          ; THIS ROUTINE IS ENTERED FROM THE QUEUE I/O DIRECTIVE WHEN AN
189          ; I/O REQUEST IS QUEUED AND AT THE END OF A PREVIOUS I/O OPERATION
190          ; TO PROPAGATE THE EXECUTION OF THE DRIVER. IF THE SPECIFIED
191          ; CONTROLLER IS NOT BUSY, THEN ATTEMPT IS MADE TO DEQUEUE THE
192          ; NEXT I/O REQUEST. ELSE A RETURN TO THE CALLER IS EXECUTED. IF
193          ; THE DEQUEUE ATTEMPT IS SUCCESSFUL, THEN THE NEXT I/O OPERATION
194          ; IS INITIATED. A RETURN TO THE CALLER IS THEN EXECUTED.
195          ;
196          ; INPUT:
197          ;
198          ; R5 = ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED
199          ;
200          ; OUTPUT:
201          ;
202          ; IF THE SPECIFIED CONTROLLER IS NOT BUSY AND AN I/O REQUEST
203          ; IS WAITING TO BE PROCESSED, THEN THE REQUEST IS DEQUEUED
204          ; AND THE DRIVER INITIATES THE REQUESTED I/O FUNCTION
205          ;
206          ;
207 000566          .ENABL LSB
208 000572 103001 MAINI: CALL $GTPKT          ;GET AN I/O PACKET TO PROCESS
209 000574          BCC 10$
210          RETURN          ;IF CS CONTROLLER BUSY OR NO REQUEST
211          ;
212          ; THE FOLLOWING ARGUMENTS ARE RETURNED BY $GTPKT:
213          ;
214          ; R1=ADDRESS OF THE I/O REQUEST PACKET.
215          ; R2=PHYSICAL UNIT NUMBER.
216          ; R3=CONTROLLER INDEX.
217          ; R4=ADDRESS OF THE STATUS CONTROL BLOCK.
218          ; R5=ADDRESS OF THE UCB OF THE CONTROLLER TO BE INITIATED.
219          ;
220          ; MAGNETIC TAPE FUNCTION INDEPENDENT I/O REQUEST PACKET FORMAT:
221          ;
222          ; I.LNK WD. 00 = I/O QUEUE THREAD WORD.
223          ; I.PRI WD. 01 = REQUEST PRIORITY (LOW BYTE).
224          ; I.EFN WD. 01 = EVENT FLAG NUMBER (HIGH BYTE).
225          ; I.TCB WD. 02 = ADDRESS OF THE TCB OF THE REQUESTOR TASK.
226          ; I.LN2 WD. 03 = POINTER TO 2ND LUN WORD IN REQUESTOR TASK HEADER.
227          ; I.UCB WD. 04 = CONTENTS OF 1ST LUN WORD IN REQUESTOR TASK HEADER (UCB)
228          ; I.FCN WD. 05 = I/O FUNCTION CODE (IO.RLB/IO.RWD/IO.SPB/ETC.).

```

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-4

```

229 ;I.IOSB WD. 06 = VIRTUAL ADDRESS OF I/O STATUS BLOCK.
230 ;IOSB+2 WD. 07 = RELOCATION BIAS OF I/O STATUS BLOCK.
231 ;IOSB+4 WD. 10 = I/O STATUS BLOCK ADDRESS (REAL OR DISPLACEMENT+140000).
232 ;I.AST WD. 11 = VIRTUAL ADDRESS OF AST SERVICE ROUTINE.
233 ;
234 ;
235 ; READ/WRITE LOGICAL FUNCTION DEPENDENT I/O PACKET FORMAT:
236 ;
237 ;I.PRM WD. 12 = PAGE ADDRESS OF DATA BUFFER ADDRESS.
238 ;PRM+2 WD. 13 = OFFSET OF DATA BUFFER ADDRESS.
239 ;PRM+4 WD. 14 = NUMBER OF BYTES TO BE TRANSFERRED.
240 ;PRM+6 WD. 15 = NOT USED.
241 ;PRM+10 WD. 16 = NOT USED.
242 ;PRM+12 WD. 17 = NOT USED.
243 ;PRM+14 WD. 20 = RELOCATION BIAS OF DIAGNOSTIC REGISTER BLOCK ADDRESS.
244 ;PRM+16 WD. 21 = DIAG. REG. BUFFER ADDRESS (REAL/DISPLACEMENT+140000).
245 ;
246 ;
247 ; BLOCK AND FILE SPACING FUNCTIONS DEPENDENT I/O PACKET FORMAT:
248 ;
249 ;I.PRM WD. 12 = SPACING COUNT (POSITIVE=FORWARD, NEGATIVE=BACKWARD).
250 ;PRM+2 WD. 13 = NOT USED.
251 ;PRM+4 WD. 14 = NOT USED.
252 ;PRM+6 WD. 15 = NOT USED.
253 ;PRM+10 WD. 16 = NOT USED.
254 ;PRM+12 WD. 17 = RELOCATION BIAS OF DIAGNOSTIC REGISTER BLOCK ADDRESS.
255 ;PRM+14 WD. 20 = DIAG. REG. BUFFER ADDRESS (REAL/DISPLACEMENT+140000).
256 ;PRM+16 WD. 21 = NOT USED.
257 ;
258 ;
259 ; (MOUNT AND) SET CHARACTERISTICS FUNCTION DEPENDENT I/O PACKET FORMAT.
260 ; FOR SENSE CHARACTERISTICS, I.PRM (WD. 12) IS NOT USED:
261 ;
262 ;I.PRM WD. 12 = NEW CHARACTERISTICS WORD.
263 ;PRM+2 WD. 13 = NOT USED.
264 ;PRM+4 WD. 14 = NOT USED.
265 ;PRM+6 WD. 15 = NOT USED.
266 ;PRM+10 WD. 16 = NOT USED.
267 ;PRM+12 WD. 17 = NOT USED.
268 ;PRM+14 WD. 20 = NOT USED.
269 ;PRM+16 WD. 21 = NOT USED.
270 ;
271 ;
272 ; WREOF, REWIND, AND UNLOAD FUNCTIONS DEPENDENT I/O PACKET FORMAT:
273 ;
274 ;I.PRM WD. 12 = NOT USED.
275 ;PRM+2 WD. 13 = NOT USED.
276 ;PRM+4 WD. 14 = NOT USED.
277 ;PRM+6 WD. 15 = NOT USED.
278 ;PRM+10 WD. 16 = NOT USED.
279 ;PRM+12 WD. 17 = RELOCATION BIAS OF DIAGNOSTIC REGISTER BLOCK ADDRESS.
280 ;PRM+14 WD. 20 = DIAG. REG. BUFFER ADDRESS (REAL/DISPLACEMENT+140000).
281 ;PRM+16 WD. 21 = NOT USED.
282 ;
283 ;
284 000576 010567 177176 10$: MOV R5, CNTBL
285 000602 012702 000506' MOV #LGFCN, R2 ;POINT TO LEGAL FUNCTION TABLE

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-5

```

286 000606 016100 000012      MOV     I.FCN(R1), R0      ;GET FUNCTION CODE
287 000612 112767 000000 177212  MOVB    #0, DPCODE      ;GET READY TO SET UP THE DPCODE
288 000620 032700 0000006      BIT     #IO.X, R0      ;ARE RETRYS SUPPRESSED?
289 000624 001403              BEQ     15$      ;IF EQ THEN NOT
290 000626 156767 000200 177176  BISB    NO.RETRY, DPCODE      ;SIGNAL SO
291 000634 132765 000004 000005 15$: BITB    #US.LAB, U.STS(R5)      ;IS THIS A LABELED TAPE?
292 000642 001403              BEQ     17$      ;IF EQ THEN NOT
293 000644 152767 000100 177160  BISB    #LAB.TAPE, DPCODE      ;SIGNAL ANSI LABELED
294 000652 042700 0000006      17$: BIC     #IO.X, R0      ;MASK OUT THE RETRY SUPPRESS BIT
295 000656 020022              20$: CMP     R0, (R2)+      ;IS IT A LEGAL FUNCTION ?
296 000660 001406              BEQ     35$      ;EQ IS YES
297 000662 062702 000002      ADD     #2, R2      ;NEXT FUNCTION CODE
298 000666 020227 000556'      CMP     R2, #ENDLGF      ;ARE WE AT THE END OF THE TABLE ?
299 000672 001371              BNE     20$      ;IF NOT THEN JUMP BACK
300 000674 000441              BR      40$      ;IF WE GET HERE THEN ILLEGAL FUNCTION
301                                ;
302 000676 016567 000024 177100 35$: MOV     U.BUF(R5), COUNT      ;SAVE THE COUNT VALUE
303 000704 142767 000202 177121  BICB    #TR.DATA!TR.DIN, TRSTATUS      ;ASSUME THAT THERE IS NO DATA
304 000712 020027 0000006      CMP     R0, #IO.WLB      ;WRITE LOGICAL FUNCTION ?
305 000716 001006              BNE     37$      ;IF NE THEN NOT
306 000720 152767 000002 177105  BISB    #TR.DATA, TRSTATUS      ;THERE IS DATA
307 000726 016567 000030 177050  MOV     U.CNT(R5), COUNT      ;THIS IS THE CORRECT COUNT VALUE
308 000734 020027 0000006      37$: CMP     R0, #IO.RLB      ;READ LOGICAL FUNCTION ?
309 000740 001006              BNE     38$      ;NE IS NOT
310 000742 152767 000200 177063  BISB    #TR.DIN, TRSTATUS
311 000750 016567 000030 177026  MOV     U.CNT(R5), COUNT      ;THIS IS THE CORRECT COUNT VALUE
312 000756 152267 177050      38$: BISB    (R2)+, DPCODE      ;ADD THE COMMAND TO THE DPCODE BYTE
313 000762 132765 000010 000005  BITB    #US.PWF, U.STS(R5)      ;IS THE POWER-FAIL BIT SET ?
314 000770 001425              BEQ     45$      ;IF EQ THEN NO
315 000772 105712              TSTB    (R2)      ;WHAT ACTION SHOULD WE TAKE ?
316 000774 100423              BMI     45$      ;IF MINUS THEN NOTHING
317 000776 001017              BNE     43$      ;IF NOT ZERO THEN RESET POWER FAIL
318 001000 012700 000000C      40$: MOV     #IE.IFC&377, R0      ;ELSE NOT A VALID DPCODE
319 001004 012701 000000      MOV     #0, R1      ;SET UP FOR IODONE
320                                ;
321                                ;
322                                ; ENTRY POINT FOR END OF FUNCTION
323                                ;
324                                ;
325 001010      IODONE: CALL    $IODON      ;GO
326 001014 000664      BR      MAINI      ;CHECK FOR MORE WORK
327                                ;
328 001016      DIAG: CALL    $FORK
329 001022 105726      DIAG1: TSTB    (SP)+
330 001024 012701 000000      MOV     #0, R1
331 001030 012700 000000C      MOV     #IE.DNR&377, R0
332 001034 000765      BR      IODONE
333                                ;
334 001036 142765 000010 000005 43$: BICB    #US.PWF, U.STS(R5)      ;RESET THE POWER FAIL BIT
335                                ;
336 001044 152767 000001 176761 45$: BISB    #TR.STR, TRSTATUS
337 001052 142765 000001 000005  BICB    #US.ABO, U.STS(R5)      ;CLEAR ANY ABORT CONDITION
338                                ;
339                                ;
340 001060 012703 000046'      MOV     #TDATA, R3      ;SET UP THE START ADDRESS OF THE FRAME BUFFER
341 001064 116723 176742      MOVB    DPCODE, (R3)+      ;SET UP THE DPCODE
342 001070 116523 000006      MOVB    U.UNIT(R5), (R3)+      ;AND THE UNIT NUMBER

```

MADR V MACRO M1200 20-JUN-86 15:43 PAGE 3-6

```

343 001074 016523 000012      MOV      U.CNT(R5), (R3)+      ;THE TAPE STATUS
344 001100 016723 176700      MOV      COUNT, (R3)+      ;THEN THE COUNT
345 001104 012702 000072      MOV      #NUMDAT-6, R2      ;NOW LOAD THE NUMBER OF BYTES PER FRAME
346 001110 142767 000004 176715  BICB     #TR.END, TRSTATUS      ;NOT THE END YET
347                                ;
348 001116                                NEXTFRM:
349 001116 012767 000005 176672      MOV      #RETRY, RETRYCNT      ;SET UP THE RETRY COUNTER
350 001124 012767 000000 176650      MOV      #0, TLEN
351 001132 116767 176704 176675      MOVB     TFNUM, FNUMOUT
352 001140 112767 000002 176672      MOVB     #START, TFCHAR
353 001146 112767 000004 176665      MOVB     #END, TTCHAR      ;ASSUME THIS IS THE END FRAME
354 001154 132767 000002 176651      BITB     #TR.DATA, TRSTATUS      ;IS THERE DATA?
355 001162 001424                        BEQ      50$      ;IF EQ THEN NO DATA
356 001164 142767 000002 176641      BICB     #TR.DATA, TRSTATUS      ;ASSUME NO MORE DATA
357 001172                                48$: CALL     #GTBYT      ;GET A DATA BYTE
358 001176 112623                        MOVB     (SP)+, (R3)+      ;INSERT THE DATA IN THE FRAME
359 001200 005365 000030                        DEC      U.CNT(R5)
360 001204 005302                        DEC      R2
361 001206 005765 000030                        TST      U.CNT(R5)      ;IS THIS THE LAST BYTE?
362 001212 001410                        BEQ      50$      ;IF EQ THEN YES
363 001214 005702                        TST      R2      ;THE LAST BYTE IN THE FRAME?
364 001216 001365                        BNE      48$
365 001220 112767 000003 176613      MOVB     #CONT, TTCHAR      ;LOAD A CONT CHARACTER
366 001226 152767 000002 176577      BISB     #TR.DATA, TRSTATUS      ;SIGNAL THAT THERE IS MORE DATA
367 001234 012767 000100 176540 50$: MOV      #NUMDATA, TLEN      ;CALCULATE THE NUMBER OF BYTES
368 001242 160267 176534                        SUB      R2, TLEN
369 001246 152767 000100 176557      BISB     #TR.ACR, TRSTATUS
370
371                                ; ENTRY POINT FOR FRAME RETRY
372                                ;
373 001254                                SENDFRAME:
374 001254 152767 000040 176551      BISB     #TR.BSY, TRSTATUS
375 001262 012767 000000 176520      MOV      #0, RINDEX      ;SET UP TO RECEIVE A FRAME
376 001270 012767 000000 176522      MOV      #0, RSTATE
377 001276 012767 177777 176520      MOV      #CRCINIT, RCRC
378 001304 016504 000020                        MOV      U.SCB(R5), R4      ;GET THE SCB ADDRESS
379 001310 116464 000007 000006      MOVB     S.ITM(R4), S.CTM(R4)      ;NOW SET UP THE TIMEOUT VALUE
380 001316 016404 000012                        MOV      S.CSR(R4), R4      ;GET THE HARDWARE ADDRESS
381 001322 012767 000000 176472      MOV      #0, TSTATE      ;SET UP TO TRANSMIT
382 001330 012767 177777 176470      MOV      #CRCINIT, TCRC
383 001336 012767 000000 176442      MOV      #0, TINDEX
384 001344 122767 000002 176466      CMPB     #START, TFCHAR      ;IS THIS A DATA FRAME?
385 001352 001011                        BNE      55$      ;IF NE THEN NOT
386 001354 005367 176436                        DEC      RETRYCNT      ;CHECK FOR MAX RETRYS
387 001360 001006                        BNE      55$      ;IF NE THEN STILL OK
388 001362 012701 000000                        MOV      #0, R1      ;IF MAX RETRYS THEN EXIT WITH TIME OUT
389 001366 012700 000000C      MOV      #IE.TMO&377, R0
390 001372 000167 177412                        JMP      IODONE
391                                ;
392 001376 032764 000200 000004 55$: BIT      #TRDY, TCS(R4)      ;READY FOR A CHARACTER?
393 001404 001403                        BEQ      60$      ;IF NOT READY WAIT
394 001406 005267 176410                        INC      TSTATE
395 001412 000427                        BR       TXDLE
396 001414 052764 000100 000004 60$: BIS      #TIE, TCS(R4)      ;WAIT FOR READY
397 001422                                RETURN
398                                ;
399                                ;

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-7

```

400
401
402 001424          ;MAOUT::
403 001424          INTSV$ MA,PR5,1
404 001430 016504 000020      MOV     U.SCB(R5), R4      ;GET SCB ADDRESS
405 001434 016404 000012      MOV     S.CSR(R4), R4      ;GET HARDWARE ADDRESS
406 001440 042764 000100 000004      BIC     #TIE, TCS(R4)
407 001446 016746 176350      MOV     TSTATE, -(SP)      ;GET READY FOR INDIRECT JUMP VIA STATE TABLE
408 001452 005267 176344      INC     TSTATE      ; AND GET READY FOR THE NEXT STATE
409 001456 006316            ASL     (SP)
410 001460 062716 000464'      ADD     #TSTTBL, (SP)
411 001464 017616 000000      MOV     @SP, (SP)
412 001470 000136            JMP     @SP+      ; AND JUMP
413
414
415 001472 112746 000020      TXOLE: MOVB   #OLE, -(SP)      ;LOAD A DLE
416 001476 000445            BR     TEXT
417
418
419 001500          TXFCHAR:
420 001500 116746 176334      MOVB   TFCHAR, -(SP)      ;LOAD THE START CHARACTER
421 001504 122767 000003 176323      CNPB   #RESETFNUM, FNUMOUT      ;DO WE WANT TO TRANSMIT A FRAME NUMBER?
422 001512 001037            BNE     TEXT      ;IF NE THEN YES
423 001514 012767 000005 176300      MOV     #5., TSTATE      ;SEND A DLE
424 001522 000433            BR     TEXT
425
426
427 001524 116746 176305      TXFNUM: MOVB   FNUMOUT, -(SP)      ;LOAD THE FRAME NUMBER
428 001530 005767 176246      TST     TLEN      ;IS THERE DATA?
429 001534 001026            BNE     TEXT      ;IF NE THEN YES
430 001536 012767 000005 176256      MOV     #5., TSTATE      ;SKIP THE DATA STATES
431 001544 000422            BR     TEXT
432
433
434 001546 016705 176234      TXDATA: MOV     TINDEX, R5      ;GET THE DATA INDEX
435 001552 116546 000046'      MOVB   TDATA(R5), -(SP)      ;LOAD IT
436 001556 122716 000020      CNPB   #OLE, (SP)      ;DO WE NEED TO STUFF A CHARACTER?
437 001562 001413            BEQ     TEXT      ;IF EQ THEN YES
438 001564 005367 176232      DEC     TSTATE      ;STAY IN THIS STATE THEN
439 001570 005205      62$: INC     R5      ;ONE MORE CHARACTER DONE
440 001572 010567 176210      MOV     R5, TINDEX      ;STORE THE UPDATED INDEX
441 001576 026705 176200      CMP     TLEN, R5      ;ARE WE FINISHED?
442 001602 001003            BNE     TEXT      ;IF LE THEN NOT
443 001604 012767 000005 176210      MOV     #5., TSTATE      ;SEND THE LAST DLE
444
445 001612 111605          TEXT: MOVB   (SP), R5
446 001614 042705 177400      BIC     #177400, R5      ;CLEAR THE HIGH BYTE
447 001620 074567 176202      XOR     R5, TCRC
448 001624 000367 176176      SWAB   TCRC
449 001630 016705 176172      MOV     TCRC, R5
450 001634 042705 170377      BIC     #170377, R5
451 001640 073527 000004      ASHC   #4, R5
452 001644 074567 176156      XOR     R5, TCRC
453 001650 016705 176152      MOV     TCRC, R5
454 001654 042705 000377      BIC     #377, R5
455 001660 073527 177773      ASHC   #-5, R5
456 001664 074567 176136      XOR     R5, TCRC

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-8

```

457 001670 073527 177771      ASHC      #7, R5
458 001674 042705 177760      BIC        #177760, R5
459 001700 074567 176122      XCR        R5, TCRC
460 001704 112605              MOV8      (SP)+, R5      ;GET THE CHARACTER
461 001706 110564 000006      MOV8      R5, TBUF(R4)    ;OUTPUT IT
462 001712 052764 000100 000004 BIS        #TIE, TCS(R4)    ;ENABLE THE INTERRUPT
463 001720              RETURN      ; AND GO
464
465
466 001722 005367 176074      TXDDE: DEC      TSTATE
467 001726 005367 176070      DEC        TSTATE
468 001732 112746 000020      MOV8      #DLE, -(SP)    ;SEND THE DLE
469 001736 016705 176044      MOV        TINDEX, R5    ;NOW GO SEE IF THAT WAS THE LAST CHARACTER
470 001742 000712      BR        62$
471
472
473 001744      TXTCHAR:
474 001744 116746 176071      MOV8      TTCHAR, -(SP)    ;LOAD THE TERM CHARACTER
475 001750 000720      BR        TEXTIT
476
477
478 001752      TXCRCLD:
479 001752 016767 176050 176050 MOV        TCRC, TCRCV      ;SAVE THE CRC
480 001760 005167 176044      COM        TCRCV          ;WE TRANSMIT THE 1'S COMP
481 001764 116746 176040      MOV8      TCRCV, -(SP)    ;THE LOW PART OF THE CRC
482 001770 000167 177616      JMP        TEXTIT
483
484
485 001774 116764 176031 000006 TXCRC: MOV8      TCRCV+1, TBUF(R4) ;OK, THAT'S IT
486 002002 052764 000100 000000 BIS        #RIE, RCS(R4)    ;ENSURE THAT WE CAN RECEIVE
487 002010 132767 000004 176015 BIT8      #TR.ENO, TRSTATUS ;IS THIS THE END?
488 002016 001004      BNE        65$                ;IF NE THEN YES
489 002020 142767 000040 176005 BIC8      #TR.BSY, TRSTATUS
490 002026      RETURN
491
492
493 002030      65$: CALL      $FORK
494 002034 016700 175754      MOV        ERRCODE, R0
495 002040 016701 175746      MOV        RETCOUNT, R1    ;SET UP FOR IO DONE
496 002044 012702 003407      MOV        #3407, R2
497 002050 000167 176734      JMP        IO DONE
498
499
500      ; *** INTERRUPT ENTRY POINT ***
501
502 002054      $MAINP::
503 002054      INTSV$ MA, PR5, 1
504 002060 016504 000020      MOV        U.SCB(R5), R4    ;GET SCB ADDRESS
505 002064 016404 000012      MOV        S.CSR(R4), R4    ;GET HARDWARE ADDRESS
506 002070 042764 000100 000000 BIC        #RIE, RCS(R4)    ;DISABLE INTERRUPTS
507 002076 010546      MOV        R5, -(SP)
508 002100 016405 000002      MOV        RBUF(R4), R5    ;GET THE INPUT CHARACTER
509 002104 100547      BMI        REXIT                ;IF ERROR THEN IGNORE
510 002106 132767 000040 175717 BIT8      #TR.BSY, TRSTATUS ;DO WE WANT THIS INTERRUPT?
511 002114 001143      BNE        REXIT                ;IF NE THEN NOT
512
513 002116 010546      MOV        R5, -(SP)

```

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-9

```

514 002120 042705 177400      BIC      #177400, R5
515 002124 074567 175674      XOR      R5, RCRC
516 002130 000367 175670      SWAB     RCRC
517 002134 016705 175664      MOV      RCRC, R5
518 002140 042705 170377      BIC      #170377, R5
519 002144 073527 000004      ASHC     #4, R5
520 002150 074567 175650      XOR      R5, RCRC
521 002154 016705 175644      MOV      RCRC, R5
522 002160 042705 000377      BIC      #377, R5
523 002164 073527 177773      ASHC     #-5, R5
524 002170 074567 175630      XOR      R5, RCRC
525 002174 073527 177771      ASHC     #-7, R5
526 002200 042705 177760      BIC      #177760, R5
527 002204 074567 175614      XOR      R5, RCRC
528 002210 012605              MOV      (SP)+, R5      ;GET THE CHARACTER BACK
529                                ;
530 002212 016746 175602      MOV      RSTATE, -(SP)      ;GET READY FOR INDIRECT JUMP VIA STATE TABLE
531 002216 005267 175576      INC      RSTATE
532 002222 006316              ASL      (SP)
533 002224 062716 000446      ADD      #RSTTBL, (SP)
534 002230 017616 000000      MOV      @ (SP), (SP)
535 002234 000136              JMP      @ (SP)+      ; AND JUMP
536                                ;
537                                ;
538                                ;
539 002236              RCFXOLE:
540 002236 142767 000010 175567      BICB     #TR.ERR, TRSTATUS      ;NO ERRORS YET!
541 002244 122705 000020              CMPB     #OLE, R5      ;OLE RECEIVED?
542 002250 001465              BEQ      REXIT      ;IF SO THEN FINE
543 002252 012767 000000 175540      MOV      #0, RSTATE      ;IF NOT THEN STAY THE WAY WE WERE
544 002260 012767 177777 175536      MOV      #CRCINIT, RCRC
545 002266 012767 000000 175514      MOV      #0, RINDEX
546 002274 000453              BR      REXIT
547                                ;
548                                ;
549                                ;
550 002276              RCFCHAR:
551 002276 110567 175534      MOVB     R5, RCFCHAR      ;STORE THE FIRST CHARACTER
552 002302 122705 000002      CMPB     #START, R5      ;NOW SEE IF IT WAS VALID
553 002306 001446              BEQ      REXIT
554 002310 122705 000006      CMPB     #ACK, R5
555 002314 001443              BEQ      REXIT
556 002316 122705 000025      CMPB     #NAK, R5
557 002322 001440              BEQ      REXIT
558 002324 122705 000004      CMPB     #END, R5      ;IF AN END OR CONT THEN ERROR, GET THE
559                                ; CRC AND SEND A NAK FRAME
560 002330 001403              BEQ      #0$
561 002332 122705 000003      CMPB     #CONT, R5
562 002336 001344              BNE      #0$
563 002340 152767 000010 175465 80$:      BISS     #TR.ERR, TRSTATUS      ;ENSURE A NAK FRAME
564 002346 012767 000005 175444      MOV      #5, RSTATE
565 002354 112767 000025 175454      MOVB     #NAK, RCFCHAR      ;ASSUME THAT WE ARE WAITING FOR AN ACK FRAME
566 002362 132767 000100 175443      BITB     #TR.ACR, TRSTATUS
567 002370 001015              BNE      REEXIT
568 002372 112767 000002 175436      MOVB     #START, RCFCHAR
569 002400 000411              BR      REEXIT
570                                ;

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-10

```

571
572 002402 122705 000000      ;
RCFNUN: CMPB    #0, R5          ;CHECK FOR VALID FRAME NUMBER
573 002406 001403      BEQ    90$
574 002410 122705 000001      CMPB    #1, R5          ;IF NEITHER OF THESE THEN DISASTER
575 002414 001315      BNE    70$
576 002416 110567 175412      90$: MOVB    R5, FNUMIN      ;STORE THE FRAME NUMBER
577 002422 000400      BR      REXIT
578
579 002424      ;
REEXIT:
580 002424 012605      REXIT: MOV    (SP)+, R5          ;RESTORE R5
581 002426 052764 000100 000000      BIS    #RIE, RCS(R4)      ;SET RECEIVE INTERRUPTS
582 002434      RETURN
583
584
585 002436 122705 000020      RCDATA: CMPB    #OLE, R5          ;WAS IT A OLE?
586 002442 001770      BEQ    REXIT          ;IF SO THEN NO ACTION TILL THE NEXT CHARACTER
587 002444 005367 175350      DEC    RSTATE          ;WANT TO STAY IN THIS STATE
588 002450 022767 000177 175332 100$: CMP    #BUFLN-1, RINDEX      ;IS THERE STILL SPACE IN THE BUFFER?
589 002456 001762      BEQ    REXIT          ;IF NO ROOM THEN IGNORE
590 002460 010446      MOV    R4, -(SP)          ;SAVE R4
591 002462 016704 175322      MOV    RINDEX, R4
592 002466 110564 000246      MOVB    R5, RDATA(R4)          ;STORE THE DATA
593 002472 005267 175312      INC    RINDEX          ;NEW INDEX
594 002476 012604      MOV    (SP)+, R4          ;RESTORE R4
595 002500 000751      BR      REXIT
596
597
598 002502 110567 175331      RCDLE: MOVB    R5, RTCHAR          ;STORE THE POSSIBLE TERMINATING CHARACTER
599 002506 122705 000004      CMPB    #ENO, R5          ;END?
600 002512 001744      BEQ    REXIT
601 002514 122705 000003      CMPB    #CONT, R5          ;CONT?
602 002520 001741      BEQ    REXIT
603 002522 122705 000020      CMPB    #OLE, R5          ;IF OLE THEN THE CHARACTER IS DATA
604 002526 001250      BNE    70$
605 002530 005367 175264      DEC    RSTATE
606 002534 005367 175260      DEC    RSTATE
607 002540 000743      BR      100$
608
609
610 002542 022767 170270 175254      RCCRC: CMP    #CRCFIN, RCRC          ;IF OK THEN THIS SHOULD BE ZERO
611 002550 001403      BEQ    110$
612 002552 152767 000010 175253      BISB    #TR.ERR, TRSTATUS          ;SIGNAL ERROR
613 002560 052764 000100 000000 110$: BIS    #RIE, RCS(R4)          ;ENABLE INTERRUPTS
614 002566 012767 000000 175224      MOV    #0, RSTATE          ;READY FOR NEXT FRAME
615 002574 152767 000040 175231      BISB    #TR.BSY, TRSTATUS          ;SIGNAL THAT THE RECEIVER IS BUSY
616 002602 012605      MOV    (SP)+, R5          ;RESTORE R5
617 002604 122767 000025 175224      CMPB    #NAK, RFCHAR          ;NAK FRAME?
618 002612 001565      BEQ    170$
619 002614 122767 000006 175214      CMPB    #ACK, RFCHAR          ;OR MAYBE AN ACK FRAME?
620 002622 001411      BEQ    120$
621 002624 132767 000100 175201      BITB    #TR.ACR, TRSTATUS
622 002632 001157      BNE    180$
623 002634 122767 000002 175174      CMPB    #START, RFCHAR          ;IS THIS A START FRAME?
624 002642 001153      BNE    180$          ;IF NOT THEN IGNORE THE FRAME
625 002644 000423      BR      130$
626 002646 132767 000010 175157 120$: BITB    #TR.ERR, TRSTATUS          ;WAS THERE AN ERROR IN THE RECEIVED FRAME?
627 002654 001146      BNE    180$          ;IF NE THEN YES

```

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-11

```

628 002656 126767 175152 175156      CMPB    FNUMIN, TFNUM      ;CHECK FRAME NUMBERS
629 002664 001142                      BNE      180$          ;IF NOT THEN JUST IGNORE
630 002666 012704 000001              MOV      #1, R4          ;IF WE GET HERE THEN RECEIVED OK
631 002672 074467 175144              XOR      R4, TFNUM        ;NEXT FRAME NUMBER
632 002676 142767 000100 175127      BICB     #TR.ACR, TRSTATUS
633 002704 132767 000002 175121      BITB     #TR.DATA, TRSTATUS ;IS THERE DATA?
634 002712 001527                      BEQ      180$          ;THEN GO WAIT FOR A RETURN FRAME
635                                     ;
636                                     ;AT THIS POINT WE HAVE COMPLETED ALL TIME CRITICAL CODE, SO FORK AND THEN SORT OUT THE REST
637                                     ;
638                                     ;
639 002714                      130$: CALL     $FORK              ;AND PROCESS AT LEISURE
640 002720 122767 000006 175110      CMPB     #ACK, RFCHAR        ;FIRST FINISH WITH THE ACK FRAME.
641                                     ; IF WE GOT HERE WITH AN ACK FRAME THEN
642                                     ; THERE WAS NO ERROR, AND WE NEED MORE DATA
643 002726 001006                      BNE      140$
644 002730 012703 000046'              MOV      #TDATA, R3
645 002734 012702 000100              MOV      #NUMDATA, R2      ;SET UP FOR THE NEXT FRAME
646 002740 000167 176152              JMP      NEXTFRM
647                                     ;
648 002744 122767 000002 175064 140$: CMPB     #START, RFCHAR      ;A START FRAME IS THE ONLY OTHER VALID
649                                     ;POSSIBILITY
650 002752 001107                      BNE      180$          ;IF NOT THEN IGNORE THE FRAME
651 002754 116767 175054 175053      MOVB     FNUMIN, FNUMOUT    ;SET UP FOR AN ACK OR NAK FRAME
652 002762 112767 000004 175051      MOVB     #END, TTCHAR
653 002770 112767 000025 175042      MOVB     #NAK, TFCHAR      ;ASSUME A NAK FRAME
654 002776 012767 000000 174776      MOV      #0, TLEN
655 003004 132767 000010 175021      BITB     #TR.ERR, TRSTATUS ;WAS THERE AN ERROR IN THE RECEIVED FRAME
656 003012 001065                      BNE      170$          ;IF YES THEN YES
657 003014 112767 000006 175016      MOVB     #ACK, TFCHAR      ;SET UP FOR AN ACK FRAME
658 003022 126767 175006 175014      CMPB     FNUMIN, RFNUM    ;CORRECT FRAME NUMBER?
659 003030 001056                      BNE      170$          ;IF NOT THEN ASSUME A LOST ACK FRAME
660 003032 012704 000001              MOV      #1, R4          ;SET THE NEW FRAME NUMBER
661 003036 074467 175002              XOR      R4, RFNUM
662 003042 012702 000246'              MOV      #RDATA, R2
663 003046 016703 174736              MOV      RINDEX, R3      ;SET UP TO GET THE DATA
664 003052 132767 000001 174753      BITB     #TR.STR, TRSTATUS ;IS THIS THE FIRST RETURN FRAME?
665 003060 001413                      BEQ      150$          ;IF EQ THEN NOT, ALL THE INFO IN THE
666                                     ; FRAME IS DATA
667 003062 012265 000012              MOV      (R2)+, U.CW2(R5) ;GET THE TAPE STATUS
668 003066 012267 174720              MOV      (R2)+, RETCOUNT ; AND THE COUNT VALUE
669 003072 012267 174716              MOV      (R2)+, ERRCODE   ; AND THE ERROR CODE
670 003076 142767 000001 174727      BICB     #TR.STR, TRSTATUS
671 003104 162703 000006              SUB      #6., R3
672                                     ;
673 003110 132767 000200 174715 150$: BITB     #TR.DIN, TRSTATUS
674 003116 001414                      BEQ      160$
675 003120 005703                      TST      R3            ;STILL DATA?
676 003122 001412                      BEQ      160$          ;IF EQ THEN NOT
677 003124 005765 000030              TST      U.CNT(R5)      ;STILL WANT DATA?
678 003130 001407                      BEQ      160$          ;IF EQ THEN NOT
679 003132 112246                      MOVB     (R2)+, -(SP)    ;SET UP
680 003134                      CALL     $PTBYT      ;INSERT THE DATA
681 003140 005365 000030              DEC      U.CNT(R5)
682 003144 005303                      DEC      R3
683 003146 000760                      BR       150$
684                                     ;

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-12

```

685 003150 122767 000004 174661 160$: CMPS  #END, RTCHAR      ;IS THIS THE END FRAME?
686 003156 001003          BNE  170$      ;IF NE THEN NOT
687 003160 152767 000004 174645          BISS  #TR.END, TRSTATUS ;IF SO THEN WANT TO CALL IODONE AFTER ACK FRAME
688 003166 000167 176062          170$: JMP  SENDFRAME
689 003172 012767 000000 174610 180$: MOV  #0, RINDEX
690 003200 012767 000000 174612          MOV  #0, RSTATE
691 003206 012767 177777 174610          MOV  #CRCINIT, RCRC
692 003214 142767 000040 174611          BICB  #TR.BSY, TRSTATUS
693 003222          RETURN
694          ;
695          ;
696          ;
697          ;
698          .DSABLE LSB
699          .ENABLE LSB
700          ;
701          ;+
702          ;***** DRIVER CANCEL ENTRY POINT *****
703          ;
704          ; THE CANCEL I/O OPERATION IS BASICALLY A NOP FOR
705          ; MAGNETIC TAPES. HOWEVER, IF TIME OUT OCCURS THEN THE
706          ; OPERATION WILL BE ABORTED RATHER THAN BEING RETRIED.
707          ;
708          ; INPUTS:
709          ; R0 = ACTIVE I/O PACKET ADDRESS
710          ; R1 = CURRENT TCB ADDRESS
711          ; R3 = CONTROLLER INDEX
712          ; R4 = SCB ADDRESS
713          ; R5 = UCB ADDRESS
714          ;-
715          ;
716          ;
717 003224 026001 000004          MACAN: CMP  I.TCB(R0), R1      ;CHECK THAT THE CANCEL IS MEANT FOR THIS TASK
718 003230 001003          BNE  5$      ;IF NE THEN NOT
719 003232 152765 000001 000005          BISS  #US.ABO, U.STS(R5) ;SET THE ABORT FLAG
720 003240          5$: RETURN
721          ;
722          ;+
723          ; **MAPWF- POWERFAIL ENTRY POINT
724          ;
725          ; POWERFAIL IS THE SAME AS INITIALIZE. OUTSTANDING REQUESTS ARE
726          ; HANDLED VIA THE DEVICE TIMEOUT FACILITY. THIS IS DONE TO AVOID
727          ; A RACE CONDITION THAT COULD EXIST IN RESTARTING THE I/C OPERATION.
728          ;
729          ; INPUTS:
730          ; R3 = CONTROLLER INDEX
731          ; R4 = SCB ADDRESS
732          ; R5 = UCB ADDRESS
733          ;-
734          ;
735          ;
736 003242          MAPWF: .IF OF P$$RFL
737          BISS  #US.PWF, U.STS(R5)      ;SET THE POWERFAIL FLAG, BUT ONLY IF
738          ; REQUESTED AT SYSGEN TIME
739          .ENDC
740 003242          RETURN
741          ;

```

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-13

```

742 ;
743 ;***** DRIVER TIME OUT ENTRY POINT *****
744 ;
745 ;
746 ; DEVICE TIMEOUT RESULTS IN A FRAME RETRY UNLESS THE MAXIMUM NUMBER
747 ; OF RETRYS HAS BEEN EXCEEDED.
748 ;
749 ; INPUTS:
750 ; R0 = LITERAL CONSTANT IE.DNR
751 ; R2 = CSR ADDRESS
752 ; R3 = CONTROLLER INDEX
753 ; R4 = SCB ADDRESS
754 ; R5 = UCB ADDRESS
755 ;
756 003244 016504 000020 MACOUT: MOV U.SCB(R5), R4 ;GET THE SCB ADDRESS
757 003250 116464 000007 000006 MOVB S.ITM(R4), S.CTM(R4) ;SET THE TIME-OUT
758 003256 016404 000012 MOV S.CSR(R4), R4 ;GET THE HARDWARE BASE ADDRESS
759 003262 042764 000100 000000 BIC #RIE, RCS(R4) ;DISABLE RECEIVE INTERRUPTS
760 003270 042764 000100 000004 BIC #TIE, TCS(R4) ;DISABLE TRANSMIT INTERRUPTS
761 003276 152767 000040 174527 BISB #TR.BSY, TRSTATUS
762 003304 MTPS #0
763 003310 132765 000001 000005 BITB #US.ABO, U.STS(R5) ;IS ABORT REQUESTED?
764 003316 001411 BEQ 10$ ;IF EQ THEN NOT
765 003320 142765 000001 000005 BICB #US.ABO, U.STS(R5) ;CLEAR THE ABORT FLAG
766 003326 012700 000000C MOV #IE.ABO&377, R0 ;TELL THE REQUESTING TASK WHAT HAPPENED
767 003332 012701 000000 MOV #0, R1
768 003336 000167 175446 JMP IDONE
769 003342 000167 175706 10$: JMP SENDFRAME
770 ;
771 ;
772 ;
773 ; .DSABLE LSB
774 ;
775 000001 .END

```


MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-14
SYMBOL TABLE

AB.NPV= 000001	BUFLN= 000200	DV.SWL= 001000	F3.EIS= 000004	I.UCB 000010
AB.TYP= 000002	B\$LV1= 031063	DV.TTY= 000004	F3.HWK= 000002	KDSAR0= 172360
ACK = 000006	B\$LV2= 020040	DV.UMD= 000200	F3.FMN= 004000	KDSOR0= 172320
AK.BUF= 000200	CNODE = 140000	D\$H11= 000001	F3.PRO= 000040	KINAR0= 172340
AK.GBI= 000202	CM.CDS= 000004	D\$LAG= 000000	F3.RLK= 020000	KINAR5= 172352
AK.GSF= 000303	CM.CEN= 000003	D\$ISK= 000000	F3.SHF= 040000	KINAR6= 172354
AK.JCB= 000201	CM.ELM= 000005	D\$LI1= 000004	F3.STM= 000010	KINAR7= 172356
AS.CAA= 000006	CM.EXT= 000006	D\$SHF= 000000	F3.TCM= 002000	KISAR0= 172340
AS.DEL= 000001	CM.IND= 000002	D\$YNC= 000000	F3.VDS= 000020	KISAR5= 172352
AS.DIS= 000002	CM.INE= 000001	D\$YNM= 000000	F3.WAT= 010000	KISAR6= 172354
AS.FPA= 000001	CM.LKT= 000007	D\$Z11= 000001	F3.XHR= 000100	KISAR7= 172356
AS.PFA= 000004	CM.MSG= 000011	END = 000004	F3.11S= 000400	KISDR0= 172300
AS.RCA= 000002	CM.RMT= 000010	ENDLGF= 000556R	GS.DEL= 000001	KISDR6= 172314
AS.REA= 000005	CNTBL 000000R	EDFCMD= 000003	G\$EFN= 000000	KISDR7= 172316
AS.RRA= 000003	CONT = 000003	ERRCDD 000014R	G\$TPP= 000000	K\$AST= 000000
A\$BID= 000000	COUNT 000004R	E\$DVC= 000000	G\$ITK= 000000	K\$CNT= 177546
A\$BRT= 000000	CP.DSB= 000010	E\$LOG= 000000	G\$WRD= 000000	K\$CSR= 177546
A\$CHK= 000000	CP.EXT= 000400	E\$XPR= 000000	G.CNT 000004	K\$IEN= 000115
A\$CLI= 000002	CP.LGD= 000004	FE.CAL= 000040	G.EFLG 000006	K\$LOC= 000001
A\$CPS= 000000	CP.MSG= 000002	FE.CEX= 020000	G.GRP 000002	K\$TPS= 000062
A\$NSI= 000000	CP.NID= 000100	FE.DRV= 000010	G.LGTH= 000012	LAB.TA= 000100
A\$PRI= 000000	CP.NUL= 000001	FE.DYM= 010000	G.LNK 000000	LD\$CD = 000000
A\$TRP= 000000	CP.PRIV= 000020	FE.EXP= 000200	G.STAT 000003	LD\$DD = 000000
A.ACC 000012	CP.RST= 000200	FE.EXT= 000001	HF.CIS= 000200	LD\$MA = 000000 6
A.ACCE 000000	CP.SGL= 000040	FE.EXV= 000004	HF.EIS= 000002	LD\$MS = 000000
A.AST 000006	CRCFIN= 170270	FE.FDT= 002000	HF.FPP= 100000	LD\$MT = 000000
A.BYT 000004	CRCINI= 177777	FE.LSI= 000400	HF.UBM= 000001	LD\$TT = 000000
A.CALL 000022	C\$CKP= 000004	FE.MUP= 000002	H\$RTZ= 000062	LGFCN 000506R
A.CBL 000002	C\$INT= 000000	FE.NXT= 040000	IE.ABD= ***** GX	L\$ASG= 000000
A.CDNW 000012	C\$ONS= 000001	FE.NLG= 100000	IE.DNR= ***** GX	L\$DRV= 000000
A.DENV 000002	C\$ORE= 002022	FE.OFF= 001000	IE.IFC= ***** GX	L\$LDR= 000000
A.DIS 000002	C\$RSH= 177564	FE.PKT= 000100	IE.TMD= ***** GX	L\$SCH= 000000
A.DISC 000014	C\$SMT= 000000	FE.PLA= 000020	IDDONE 001010R	MACAN 003224R
A.DQSR 177776	C\$TTY= 177564	FE.X25= 004000	ID.EOF= ***** GX	MAINI 000566R
A.IBUF 000012	C.PCPL 000011	FNUMIN 000034R	ID.RLB= ***** GX	MAOUT 003244R
A.ILEN 000014	C.PDPL 000010	FNUMCU 000035R	ID.RWD= ***** GX	MAPWF 003242R
A.IMAP 000010	C.PNAM 000002	F\$LPP= 000000	ID.RWU= ***** GX	MPAR = 172100
A.INPU 000006	C.PRMT 000012	F\$LVL= 000001	ID.SEC= ***** GX	MPCSR = 177746
A.IDS 000024	C.PSTS 000006	F2.ACN= 000020	ID.SMD= ***** GX	M\$CRB= 000124
A.KSR5 177774	C.PTCB 000000	F2.AHR= 010000	ID.SPB= ***** GX	M\$CRX= 000000
A.LEN1= 000014	DIAG 001016R	F2.DAS= 000001	ID.SPF= ***** GX	M\$EIS= 000000
A.LEN2= 000032	DIAG1 001022R	F2.DPR= 000400	ID.STC= ***** GX	M\$FCS= 000000
A.LIN 000010	DLE = 000020	F2.EVT= 000010	ID.WLB= ***** GX	M\$MGE= 000000
A.MAS 000004	DV.CCL= 000002	F2.GGF= 002000	IE.Y = ***** GX	M\$MUP= 000000
A.NPR 000010	DV.COM= 020000	F2.IRR= 001000	I\$RAR= 000000	M\$QVR= 000000
A.NUM 000006	DV.DIR= 000010	F2.LIB= 000002	I\$RDN= 000000	M.BFVH 000011
A.OUTPUT 000010	DV.EXT= 000400	F2.MP = 000004	I.AST 000022	M.BFVL 000012
A.POWE 000004	DV.F11= 040000	F2.POL= 000100	I.ATTL= 000044	M.LGTH= 000014
A.PRM 000012	DV.ISP= 002000	F2.RAS= 004000	I.EFN 000003	M.LNK 000000
A.PROC 000020	DV.MBC= 000400	F2.RBN= 020000	I.FCN 000012	M.UMRA 000002
A.RECE 000016	DV.MNT= 100000	F2.SDW= 000040	I.IQSB 000014	M.UMRN 000004
A.REL 000000	DV.MSD= 000100	F2.STP= 100000	I.LGTH= 000044	M.UMVH 000010
A.RES 000030	DV.OSP= 004000	F2.SWP= 040000	I.LNK 000000	M.UMVL 000006
A.SBUF 000020	DV.PSE= 010000	F2.WND= 000200	I.LN2 000006	NAK = 000025
A.SLEN 000022	DV.REC= 000001	F3.AST= 000200	I.PRI 000002	NEXTFR 001116R
A.SMAP 000016	DV.SDI= 000020	F3.CLI= 001000	I.PRM 000024	NO.RET= 000200
A.STA 000013	DV.SQD= 000040	F3.CRA= 000001	I.TCB 000004	NUMDAT= 000100

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-15
SYMBOL TABLE

N\$LDV= 000001	RETRY= 000005	S.PRI 000004	TINDEX 000006R	UA.CCM= 000200
N\$MOV= 000041	REXIT 002424R	S.RCNT 177772	TLEN 000002R	UA.ECH= 000004
N\$UMR= 000034	RFCHAR 000036R	S.ROFF 177773	TPS = 177564	UA.PRO= 000002
OPCODE 000032R	RFNUM 000044R	S.STS 000011	TRDY = 000200	UA.PUT= 000040
O.AST 000006	RIE = 000100	S.VCT 000005	TRSTAT 000033R	UA.SPE= 000020
O.EFN 000010	RINDEX 000010R	S1.BEL= 000400	TR.ACR= 000100	UA.TRA= 001000
O.ESB 000012	RLBCMD= 000001	S1.CTG= 000040	TR.BSY= 000040	UA.TYP= 000010
O.LGTH= 000034	RSTATE 000020R	S1.CTS= 010000	TR.DAT= 000002	UBMPR = 170200
O.LNK 000000	RSTTBL 000446R	S1.DEC= 002000	TR.DIN= 000200	UC.ALG= 000200
O.MCRL 000002	RTCHAR 000037R	S1.DPR= 001000	TR.END= 000004	UC.ATT= 000010
O.PTCS 000004	RWDCMD= 000004	S1.OSI= 004000	TR.ERR= 000010	UC.KIL= 000004
O.STAT 000014	RWUCMD= 000005	S1.ESC= 000004	TR.OP = 000020	UC.LSH= 000003
PC.AL= 000004	R\$EXV= 000000	S1.IBF= 100000	TR.STR= 000001	UC.NPR= 000100
PC.ALH= 001000	R\$LKL= 000001	S1.IBY= 000200	TSTATE 000022R	UC.PWF= 000020
PC.HIH= 000001	R\$LL1= 000001	S1.OBF= 040000	TSTTBL 000464R	UC.QUE= 000040
PC.LOW= 000002	R\$NDC= 000012	S1.OBY= 000100	TTCHAR 000041R	UDSAR0= 177660
PC.NRM= 000400	R\$NDH= 000144	S1.RAL= 000010	TXCRC 001774R	UDSDR0= 177620
PC.XAF= 000010	R\$NDL= 000012	S1.RNE= 000020	TXCRCL 001752R	UD.UNG= 000000
PC.XIT= 000200	R\$POI= 000000	S1.RST= 000001	TXDATA 001546R	UD.160= 000004
PF.ALL= 177777	R\$SND= 000000	S1.RUB= 000002	TXDDLE 001722R	UD.200= 000001
PF.INS= 000040	R\$TPR= 031462	S1.USI= 020000	TXDLE 001472R	UD.556= 000002
PF.LDG= 000100	R\$11N= 000000	S2.ACR= 000001	TXFCHA 001500R	UD.625= 000005
PF.RE= 000200	SECCMD= 000011	S2.BRQ= 000020	TXFNUM 001524R	UD.800= 000003
PIRQ = 177772	SENDFR 001254R	S2.CR = 000010	TXTCHA 001744R	UISAR0= 177640
PMODE = 030000	SISDR0= 172200	S2.FDX= 100000	T\$ACR= 000000	UISAR4= 177650
PRO = 000000	SMOCMD= 000012	S2.FLF= 040000	T\$BTW= 000000	UISAR5= 177652
PR1 = 000040	SPARE = 000010	S2.HFF= 020000	T\$BUF= 000000	UISAR6= 177654
PR4 = 000200	SPBCMD= 000006	S2.HFL= 003400	T\$CCA= 000000	UISAR7= 177656
PR5 = 000240	SPFCMD= 000007	S2.HHT= 010000	T\$CCO= 000000	UISDR0= 177600
PR6 = 000300	SP.EIP= 000001	S2.IRQ= 000200	T\$CPW= 000000	UISDR4= 177610
PR7 = 000340	SP.ENG= 000002	S2.ORG= 000100	T\$CTR= 000000	UISDR5= 177612
PS = 177776	SP.LDG= 000004	S2.SRQ= 000040	T\$CUP= 000000	UISDR6= 177614
P\$BPR= 000063	SRO = 177572	S2.VFL= 004000	T\$ESC= 000000	UISDR7= 177616
P\$CTL= 000000	SR3 = 172516	S2.WRA= 000006	T\$GMC= 000000	UM.CLI= 000036
P\$FRS= 000310	START = 000002	S2.WRB= 000002	T\$GTS= 000000	UM.OSB= 000200
P\$HIL= 003100	STCCMD= 000010	S3.ABD= 001000	T\$HFF= 000000	UM.NBR= 000400
P\$LAS= 000000	SWR = 177570	S3.ABP= 002000	T\$HLD= 000000	UM.JVR= 000001
P\$LDL= 001130	S\$HFC= 000036	S3.BCC= 020000	T\$KMG= 000000	US.ABQ= 000001
P\$NIC= 177564	S\$NM1= 041525	S3.DAQ= 040000	T\$LWC= 000000	US.BSP= 000002
P\$OFF= 000000	S\$NM2= 026524	S3.PCU= 100000	T\$M11= 000001	US.BSY= 000200
P\$WRD= 000000	S\$NM3= 020111	S3.RAL= 000010	T\$RED= 000000	US.FDR= 000040
Q\$OPT= 000017	S\$TIM= 000000	S3.RCU= 000400	T\$RNE= 000000	US.FRK= 000002
RBUF = 000002	S\$TDP= 000000	S3.RPD= 000020	T\$RPR= 000000	US.KPF= 000001
RCCRC 002542R	S\$WLK= 000000	S3.TAB= 000100	T\$RST= 000000	US.LAB= 000004
RCDATA 002436R	S\$WPC= 000024	S3.VER= 010000	T\$RUB= 000000	US.MDE= 000002
RCDLE 002502R	S\$WPR= 000005	S3.WAL= 004000	T\$SMC= 000000	US.MDM= 000020
RCFCHA 002276R	S\$YSZ= 004000	S3.WES= 000040	T\$SYN= 000000	US.MNT= 000100
RCFNUM 002402R	S.BMSK 177776	S3.BBC= 000200	T\$S11= 000001	US.DFL= 000001
RCFXDL 002236R	S.BMSV 177774	TBUF = 000006	T\$TRW= 000000	US.PUB= 000004
RCRC 000024R	S.CON 000010	TCRC 000026R	T\$UTB= 000000	US.PWF= 000010
RCS = 000000	S.CSR 000012	TCRCV 000030R	T\$UTQ= 000170	US.RED= 000002
RDATA 000246R	S.CTM 000006	TCS = 000004	T\$U58= 000001	US.SHR= 000001
REEXIT 002424R	S.DMCS 000020	TDATA 000046R	T\$VBF= 000000	US.SPU= 000002
RESET = 000027	S.FRK 000016	TEXT 001612R	T\$30P= 000000	US.UMD= 000010
RESETF= 000003	S.ITM 000007	TFCHAR 000040R	UA.ACC= 000001	US.VV = 000001
RETCOU 000012R	S.LHD 000000	TFNUM 000042R	UA.ALL= 000400	US.WCK= 000010
RETRYC 000016R	S.PKT 000014	TIE = 000100	UA.CAL= 000100	U\$MHI= 000000

MADRV MACRO M1200 20-JUN-86 15:43 PAGE 3-16
SYMBOL TABLE

U\$MLO= 160000	U.CM4 000016	U.RED2 000034	U.UNFL= 000052	U2.PRIV= 000010
U\$MRN= 170234	U.CYL = 000104	U.SCB 000020	U.UNIT 000006	U2.RMT= 020000
U.AC3 000052	U.DCB 000000	U.SHST= 000076	U.UNSZ= 000114	U2.R04= 100000
U.ACP = 000032	U.FCDE= 000042	U.SHUN= 000074	U.UNTI= 000060	U2.SCS= 000004
U.ADMA 000056	U.FNUM= 000040	U.SPC = 000036	U.UTMO= 000036	U2.SLV= 000200
U.AFL6 000054	U.GRP = 000102	U.STS 000005	U.VCB = 000034	U2.VT5= 000002
U.ATT 000022	U.HSTI= 000054	U.ST2 000007	U.VSER= 000120	U2.7CH= 010000
U.BPKT= 000044	U.KCSR= 000032	U.SUB = 000036	U2.A7.= 000020	U3.UPC= 020000
U.BUF 000024	U.KCS6= 000034	U.TCHP 000042	U2.CRT= 002000	U4.CR = 000100
U.CBF = 000032	U.LHD = 000040	U.TCVP 000043	U2.DH1= 100000	V\$CTR= 000400
U.CLI 177772	U.LUIC 177774	U.TFLK 000040	U2.DJ1= 040000	V\$RSN= 000040
U.CMST= 000126	U.MEDI= 000070	U.TFRQ 000037	U2.DZ1= 000100	WLBCND= 000002
U.CNT 000030	U.MLUN= 000050	U.TLPP 000036	U2.ESC= 001000	\$FDRK = ***** GX
U.COT4 000030	U.MUP 177772	U.TMTI 000047	U2.HFF= 010000	\$GTSYT= ***** GX
U.CTCB 000026	U.OTRF= 000124	U.TRCK= 000100	U2.HLD= 000040	\$GTPKT= ***** GX
U.CTL 000004	U.OWN 177776	U.TSTA 000026	U2.LDE= 000400	\$IDON= ***** GX
U.CTYP 000050	U.RBNS= 000112	U.TTAB 000034	U2.LWC= 000001	\$MAINP 002054RG
U.CW1 000010	U.RCTC= 000113	U.TTYP 000046	U2.L3S= 000004	\$MACUT 001424RG
U.CW2 000012	U.RCTS= 000110	U.TUX 000024	U2.L8S= 010000	\$MATBL 000556RG
U.CW3 000014	U.RED 000002	U.UIC 000044	U2.NEC= 004000	\$PTBYT= ***** GX

. ABS. 177776 000
003346 001

ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 13001 WORDS (51 PAGES)

DYNAMIC MEMORY: 13780 WORDS (53 PAGES)

ELAPSED TIME: 00:00:50

MADRV,MADRV/-SP=LB:C1,13EXENC/ML,LB:C11,10JRSXMC,SY:C100,34JMADRV

MATBL MACRO M1200 20-JUN-86 16:15 PAGE 3

```

1      .TITLE  MATBL
2      .IDENT  /MA.1/
3
4
5      ; THIS IS THE DATA TABLE FOR MADRV, BASED ON THAT FROM MTDV.;
6
7      .MCALL  HWDDF$, SCBDF$, UCBD$
8      000000  HWDDF$      ; DEFINE HARDWARE OFFSETS
9      000000  SCBDF$
10     000000  UCBD$
11
12     ;
13     ;
14     000000  $MATBL = 0
15     000000  $MADAT::
16     000002  000040'  MADCB: .WORD 0
17     000004      115    101    .WORD .MAO
18     000006      000    001    .ASCII /MA/
19     000010  000046    .BYTE 0,1
20     000012  000000    .WORD MAND-MAST
21     .IF OF A$NSI
22     000014  171177 000170 010000 .WORD 171177,170,10000,161000,537,0,0,537
23     000022  161000 000537 000000
24     000030  000000 000537
25
26     .IFF
27     .WORD 120177,170,160000,0,7,0,1,6
28     .ENDC
29     .WORD 0
30
31     ;
32     ;
33     ;
34     ;
35     ;
36     ;
37     ;
38     ;
39     ;
40     000034'  NAST=.
41     .WORD 0,0 ; I/O COUNT
42     .BYTE 8,5 ; S ERROR LIMIT, H ERROR LIMIT
43     .BYTE 0,0 ; S ERROR COUNT, H ERROR COUNT
44     .WORD 0
45     .WORD 0
46     .WORD 0
47     .MAO::
48     .WORD MADCB
49     .WORD -2
50     000044      221    100    .BYTE UC.AL6!UC.PWF!1,US.MNT
51     000046      000    000    .BYTE 0,0
52     000050  140141    .WORD DV.REC!DV.S&0!DV.MSD!DV.MNT!DV.F11
53     000052  000000    .WORD 0
54     000054      003    000    .BYTE UD.800, UD.UNS
55     000056  001000    .WORD 512.
56     000060  000150'    .WORD MASCB
57     000062  000000 000000 000000 .WORD 0,0,0,0,0,0,0,0
58     000070  000000 000000 000000
59     000076  000000 000000
60
61     ;
62     ;
63     ;
64     ;
65     ;
66     ;
67     ;
68     ;
69     ;
70     ;
71     ;
72     ;
73     ;
74     ;
75     ;
76     ;
77     ;
78     ;
79     ;
80     ;
81     ;
82     ;
83     ;
84     ;
85     ;
86     ;
87     ;
88     ;
89     ;
90     ;
91     ;
92     ;
93     ;
94     ;
95     ;
96     ;
97     ;
98     ;
99     ;
100    ;
101    ;
102    ;
103    ;
104    ;
105    ;
106    ;
107    ;
108    ;
109    ;
110    ;
111    ;
112    ;
113    ;
114    ;
115    ;
116    ;
117    ;
118    ;
119    ;
120    ;
121    ;
122    ;
123    ;
124    ;
125    ;
126    ;
127    ;
128    ;
129    ;
130    ;
131    ;
132    ;
133    ;
134    ;
135    ;
136    ;
137    ;
138    ;
139    ;
140    ;
141    ;
142    ;
143    ;
144    ;
145    ;
146    ;
147    ;
148    ;
149    ;
150    ;
151    ;
152    ;
153    ;
154    ;
155    ;
156    ;
157    ;
158    ;
159    ;
160    ;
161    ;
162    ;
163    ;
164    ;
165    ;
166    ;
167    ;
168    ;
169    ;
170    ;
171    ;
172    ;
173    ;
174    ;
175    ;
176    ;
177    ;
178    ;
179    ;
180    ;
181    ;
182    ;
183    ;
184    ;
185    ;
186    ;
187    ;
188    ;
189    ;
190    ;
191    ;
192    ;
193    ;
194    ;
195    ;
196    ;
197    ;
198    ;
199    ;
200    ;
201    ;
202    ;
203    ;
204    ;
205    ;
206    ;
207    ;
208    ;
209    ;
210    ;
211    ;
212    ;
213    ;
214    ;
215    ;
216    ;
217    ;
218    ;
219    ;
220    ;
221    ;
222    ;
223    ;
224    ;
225    ;
226    ;
227    ;
228    ;
229    ;
230    ;
231    ;
232    ;
233    ;
234    ;
235    ;
236    ;
237    ;
238    ;
239    ;
240    ;
241    ;
242    ;
243    ;
244    ;
245    ;
246    ;
247    ;
248    ;
249    ;
250    ;
251    ;
252    ;
253    ;
254    ;
255    ;
256    ;
257    ;
258    ;
259    ;
260    ;
261    ;
262    ;
263    ;
264    ;
265    ;
266    ;
267    ;
268    ;
269    ;
270    ;
271    ;
272    ;
273    ;
274    ;
275    ;
276    ;
277    ;
278    ;
279    ;
280    ;
281    ;
282    ;
283    ;
284    ;
285    ;
286    ;
287    ;
288    ;
289    ;
290    ;
291    ;
292    ;
293    ;
294    ;
295    ;
296    ;
297    ;
298    ;
299    ;
300    ;
301    ;
302    ;
303    ;
304    ;
305    ;
306    ;
307    ;
308    ;
309    ;
310    ;
311    ;
312    ;
313    ;
314    ;
315    ;
316    ;
317    ;
318    ;
319    ;
320    ;
321    ;
322    ;
323    ;
324    ;
325    ;
326    ;
327    ;
328    ;
329    ;
330    ;
331    ;
332    ;
333    ;
334    ;
335    ;
336    ;
337    ;
338    ;
339    ;
340    ;
341    ;
342    ;
343    ;
344    ;
345    ;
346    ;
347    ;
348    ;
349    ;
350    ;
351    ;
352    ;
353    ;
354    ;
355    ;
356    ;
357    ;
358    ;
359    ;
360    ;
361    ;
362    ;
363    ;
364    ;
365    ;
366    ;
367    ;
368    ;
369    ;
370    ;
371    ;
372    ;
373    ;
374    ;
375    ;
376    ;
377    ;
378    ;
379    ;
380    ;
381    ;
382    ;
383    ;
384    ;
385    ;
386    ;
387    ;
388    ;
389    ;
390    ;
391    ;
392    ;
393    ;
394    ;
395    ;
396    ;
397    ;
398    ;
399    ;
400    ;
401    ;
402    ;
403    ;
404    ;
405    ;
406    ;
407    ;
408    ;
409    ;
410    ;
411    ;
412    ;
413    ;
414    ;
415    ;
416    ;
417    ;
418    ;
419    ;
420    ;
421    ;
422    ;
423    ;
424    ;
425    ;
426    ;
427    ;
428    ;
429    ;
430    ;
431    ;
432    ;
433    ;
434    ;
435    ;
436    ;
437    ;
438    ;
439    ;
440    ;
441    ;
442    ;
443    ;
444    ;
445    ;
446    ;
447    ;
448    ;
449    ;
450    ;
451    ;
452    ;
453    ;
454    ;
455    ;
456    ;
457    ;
458    ;
459    ;
460    ;
461    ;
462    ;
463    ;
464    ;
465    ;
466    ;
467    ;
468    ;
469    ;
470    ;
471    ;
472    ;
473    ;
474    ;
475    ;
476    ;
477    ;
478    ;
479    ;
480    ;
481    ;
482    ;
483    ;
484    ;
485    ;
486    ;
487    ;
488    ;
489    ;
490    ;
491    ;
492    ;
493    ;
494    ;
495    ;
496    ;
497    ;
498    ;
499    ;
500    ;
501    ;
502    ;
503    ;
504    ;
505    ;
506    ;
507    ;
508    ;
509    ;
510    ;
511    ;
512    ;
513    ;
514    ;
515    ;
516    ;
517    ;
518    ;
519    ;
520    ;
521    ;
522    ;
523    ;
524    ;
525    ;
526    ;
527    ;
528    ;
529    ;
530    ;
531    ;
532    ;
533    ;
534    ;
535    ;
536    ;
537    ;
538    ;
539    ;
540    ;
541    ;
542    ;
543    ;
544    ;
545    ;
546    ;
547    ;
548    ;
549    ;
550    ;
551    ;
552    ;
553    ;
554    ;
555    ;
556    ;
557    ;
558    ;
559    ;
560    ;
561    ;
562    ;
563    ;
564    ;
565    ;
566    ;
567    ;
568    ;
569    ;
570    ;
571    ;
572    ;
573    ;
574    ;
575    ;
576    ;
577    ;
578    ;
579    ;
580    ;
581    ;
582    ;
583    ;
584    ;
585    ;
586    ;
587    ;
588    ;
589    ;
590    ;
591    ;
592    ;
593    ;
594    ;
595    ;
596    ;
597    ;
598    ;
599    ;
600    ;
601    ;
602    ;
603    ;
604    ;
605    ;
606    ;
607    ;
608    ;
609    ;
610    ;
611    ;
612    ;
613    ;
614    ;
615    ;
616    ;
617    ;
618    ;
619    ;
620    ;
621    ;
622    ;
623    ;
624    ;
625    ;
626    ;
627    ;
628    ;
629    ;
630    ;
631    ;
632    ;
633    ;
634    ;
635    ;
636    ;
637    ;
638    ;
639    ;
640    ;
641    ;
642    ;
643    ;
644    ;
645    ;
646    ;
647    ;
648    ;
649    ;
650    ;
651    ;
652    ;
653    ;
654    ;
655    ;
656    ;
657    ;
658    ;
659    ;
660    ;
661    ;
662    ;
663    ;
664    ;
665    ;
666    ;
667    ;
668    ;
669    ;
670    ;
671    ;
672    ;
673    ;
674    ;
675    ;
676    ;
677    ;
678    ;
679    ;
680    ;
681    ;
682    ;
683    ;
684    ;
685    ;
686    ;
687    ;
688    ;
689    ;
690    ;
691    ;
692    ;
693    ;
694    ;
695    ;
696    ;
697    ;
698    ;
699    ;
700    ;
701    ;
702    ;
703    ;
704    ;
705    ;
706    ;
707    ;
708    ;
709    ;
710    ;
711    ;
712    ;
713    ;
714    ;
715    ;
716    ;
717    ;
718    ;
719    ;
720    ;
721    ;
722    ;
723    ;
724    ;
725    ;
726    ;
727    ;
728    ;
729    ;
730    ;
731    ;
732    ;
733    ;
734    ;
735    ;
736    ;
737    ;
738    ;
739    ;
740    ;
741    ;
742    ;
743    ;
744    ;
745    ;
746    ;
747    ;
748    ;
749    ;
750    ;
751    ;
752    ;
753    ;
754    ;
755    ;
756    ;
757    ;
758    ;
759    ;
760    ;
761    ;
762    ;
763    ;
764    ;
765    ;
766    ;
767    ;
768    ;
769    ;
770    ;
771    ;
772    ;
773    ;
774    ;
775    ;
776    ;
777    ;
778    ;
779    ;
780    ;
781    ;
782    ;
783    ;
784    ;
785    ;
786    ;
787    ;
788    ;
789    ;
790    ;
791    ;
792    ;
793    ;
794    ;
795    ;
796    ;
797    ;
798    ;
799    ;
800    ;
801    ;
802    ;
803    ;
804    ;
805    ;
806    ;
807    ;
808    ;
809    ;
810    ;
811    ;
812    ;
813    ;
814    ;
815    ;
816    ;
817    ;
818    ;
819    ;
820    ;
821    ;
822    ;
823    ;
824    ;
825    ;
826    ;
827    ;
828    ;
829    ;
830    ;
831    ;
832    ;
833    ;
834    ;
835    ;
836    ;
837    ;
838    ;
839    ;
840    ;
841    ;
842    ;
843    ;
844    ;
845    ;
846    ;
847    ;
848    ;
849    ;
850    ;
851    ;
852    ;
853    ;
854    ;
855    ;
856    ;
857    ;
858    ;
859    ;
860    ;
861    ;
862    ;
863    ;
864    ;
865    ;
866    ;
867    ;
868    ;
869    ;
870    ;
871    ;
872    ;
873    ;
874    ;
875    ;
876    ;
877    ;
878    ;
879    ;
880    ;
881    ;
882    ;
883    ;
884    ;
885    ;
886    ;
887    ;
888    ;
889    ;
890    ;
891    ;
892    ;
893    ;
894    ;
895    ;
896    ;
897    ;
898    ;
899    ;
900    ;
901    ;
902    ;
903    ;
904    ;
905    ;
906    ;
907    ;
908    ;
909    ;
910    ;
911    ;
912    ;
913    ;
914    ;
915    ;
916    ;
917    ;
918    ;
919    ;
920    ;
921    ;
922    ;
923    ;
924    ;
925    ;
926    ;
927    ;
928    ;
929    ;
930    ;
931    ;
932    ;
933    ;
934    ;
935    ;
936    ;
937    ;
938    ;
939    ;
940    ;
941    ;
942    ;
943    ;
944    ;
945    ;
946    ;
947    ;
948    ;
949    ;
950    ;
951    ;
952    ;
953    ;
954    ;
955    ;

```

MATBL MACRO M1200 20-JUN-86 16:15 PAGE 3-1

```

54      ;      .WORD 0
55 000102 000000      .WORD 0
56 000104 000000      .WORD 0
57 000106      .MA1::
58 000106 000000'      .WORD MADC3
59 000110 000106'      .WORD -2
60 000112 221 100      .BYTE UC.AL6!UC.PWF!1,US.MNT
61 000114 001 000      .BYTE 1,0
62 000116 140141      .WORD DV.REC!DV.SQD!DV.HSD!DV.MNT!DV.F11
63 000120 000000      .WORD 0
64 000122 003 000      .BYTE UD.300,UD.UNS
65 000124 001000      .WORD 512.
66 000126 000150'      .WORD MASC8
67 000130 000000 000000 000000      .WORD 0,0,0,0,0,0,0
    000136 000000 000000 000000
    000144 000000 000000

68
69      ;
70      ;
71      ;
72      ;      .BLKW 3
73 000150      $MA0::
74 000150      MASC8:
75 000150 000000 000150'      .WORD 0,-2
76 000154 240 072      .BYTE PR5,350/4
77 000156 000 024      .BYTE 0,20.
78 000160 000 000      .BYTE 0*2,0
79 000162 176550      .WORD 176550
80 000164 000000      .WORD 0
81 000166 000000 000000 000000      .WORD 0,0,0,0
    000174 000000
82 000176 000000      .WORD 0
83 000200      $MAEN0::
84      ;
85      000001      .END

```


MATBL MACRO M1200 20-JUN-86 16:15 PAGE 3-2
SYMBOL TABLE

A\$BID= 000000	FE.NLG= 100000	KISAR6= 172354	P\$LAS= 000000	S1.IBY= 000200
A\$BRT= 000000	FE.GFF= 001000	KISAR7= 172356	P\$LQL= 001130	S1.QBF= 040000
A\$CHK= 000000	FE.PKT= 000100	KISDR0= 172300	P\$NIC= 177564	S1.QBY= 000100
A\$CLI= 000002	FE.PLA= 000020	KISDR6= 172314	P\$OFF= 000000	S1.RAL= 000010
A\$CPS= 000000	FE.X25= 004000	KISDR7= 172316	P\$WRD= 000000	S1.RNE= 000320
A\$NSI= 000000	F\$LPP= 000000	K\$AST= 000000	Q\$OPT= 000017	S1.RST= 000001
A\$PRI= 000000	F\$LVL= 000001	K\$CNT= 177546	R\$EXV= 000000	S1.RUB= 000302
A\$TRP= 000000	F2.ACN= 000020	K\$CSR= 177546	R\$LKL= 000001	S1.USH= 020000
B\$LV1= 031063	F2.AHR= 010000	K\$IEN= 000115	R\$L11= 000001	S2.ACR= 000001
B\$LV2= 020040	F2.DAS= 000001	K\$LDC= 000001	R\$NDC= 000012	S2.BRQ= 000020
CMODE = 140000	F2.DPR= 000400	K\$TPS= 000062	R\$NDH= 000144	S2.CR = 000010
C\$CKP= 000004	F2.EVT= 000010	LD\$CO = 000000	R\$NDL= 000012	S2.FDX= 100000
C\$INT= 000000	F2.GGF= 002000	LD\$DD = 000000	R\$POI= 000000	S2.FLF= 040000
C\$ONS= 000001	F2.IRR= 001000	LD\$MS = 000000	R\$SND= 000000	S2.HFF= 020000
C\$ORE= 002022	F2.LIB= 000002	LD\$MT = 000000	R\$TPR= 031462	S2.HFL= 003400
C\$RSH= 177564	F2.MP = 000004	LD\$TT = 000000	R\$11M= 000000	S2.HNT= 010000
C\$SMT= 000000	F2.POL= 000100	L\$ASG= 000000	SISDR0= 172200	S2.IRQ= 000200
C\$TTY= 177564	F2.RAS= 004000	L\$DRV= 000000	SPARE = 000010	S2.DRQ= 000100
DV.CCL= 000002	F2.RBN= 020000	L\$LDR= 000000	SP.EIP= 000001	S2.SRQ= 000040
DV.CDM= 020000	F2.SDW= 000040	L\$5OH= 000000	SP.ENB= 000002	S2.VFL= 004000
DV.DIR= 000010	F2.STP= 100000	MADCB = 000000R	SP.LOG= 000004	S2.WRA= 000006
DV.EXT= 000400	F2.SWP= 040000	MAND = 000102R	SRO = 177572	S2.WRB= 000002
DV.F11= 040000	F2.WND= 000200	MASCB = 000150R	SRJ = 172516	S3.ABD= 001000
DV.ISP= 002000	F3.AST= 000200	MAST = 000034R	SWR = 177570	S3.ABP= 002000
DV.MBC= 000400	F3.CLI= 001000	MPAR = 172100	S\$HFC= 000036	S3.BCC= 020000
DV.MNT= 100000	F3.CRA= 000001	MPCSR = 177746	S\$NM1= 041525	S3.DAC= 040000
DV.MSD= 000100	F3.EIS= 000004	M\$CRB= 000124	S\$NM2= 026524	S3.PCU= 100000
DV.DSP= 004000	F3.NWK= 000002	M\$CRX= 000000	S\$NM3= 020111	S3.RAL= 000010
DV.PSE= 010000	F3.PMN= 004000	M\$EIS= 000000	S\$TIN= 000000	S3.RCU= 000400
DV.REC= 000001	F3.PRO= 000040	M\$FCS= 000000	S\$TOP= 000000	S3.RPD= 000020
DV.SDI= 000020	F3.RLK= 020000	M\$MGE= 000000	S\$WLK= 000000	S3.TAB= 000100
DV.SGD= 000040	F3.SHF= 040000	M\$MUP= 000000	S\$WPC= 000024	S3.VER= 010000
DV.SWL= 001000	F3.STM= 000010	M\$QVR= 000000	S\$WPR= 000005	S3.WAL= 004000
DV.TTY= 000004	F3.TCM= 002000	M.BFVN 000011	S\$YSZ= 004000	S3.WES= 000040
DV.UMD= 000200	F3.UDS= 000020	M.BFVL 000012	S.BMSK 177776	S3.8BC= 000200
D\$H11= 000001	F3.WAT= 010000	M.LGTH= 000014	S.BMSV 177774	TPS = 177564
D\$IAG= 000000	F3.XHR= 000100	M.LNK = 000000	S.CON = 000010	T\$ACR= 000000
D\$ISK= 000000	F3.11S= 000400	M.UMRA 000002	S.CSR = 000012	T\$BTW= 000000
D\$L11= 000004	G\$EFN= 000000	M.UMRN 000004	S.CTM = 000006	T\$BUF= 000000
D\$SHF= 000000	G\$TPP= 000000	M.UMVH 000010	S.DMCS 000020	T\$CCA= 000000
D\$YNC= 000000	G\$TTK= 000000	M.UMVL 000006	S.FRK = 000016	T\$CCO= 000000
D\$YNN= 000000	G\$WRD= 000000	N\$LDV= 000001	S.ITM = 000007	T\$CPW= 000000
D\$Z11= 000001	HF.CIS= 000200	N\$MOV= 000041	S.LHD = 000000	T\$CTR= 000000
E\$DVC= 000000	HF.EIS= 000002	N\$UMR= 000034	S.PKT = 000014	T\$CUP= 000000
E\$LOG= 000000	HF.FPP= 100000	PIRQ = 177772	S.PRI = 000004	T\$ESC= 000000
E\$XPR= 000000	H\$RTZ= 000062	PNODE = 030000	S.RCNT 177772	T\$GMC= 000000
FE.CAL= 000040	I\$RAR= 000000	PRO = 000000	S.ROFF 177773	T\$GTS= 000000
FE.CEX= 020000	I\$RDN= 000000	PR1 = 000040	S.STS = 000011	T\$HFF= 000000
FE.DRV= 000010	KOSAR0= 172360	PR4 = 000200	S.VCT = 000005	T\$HLD= 000000
FE.DYN= 010000	KOSDR0= 172320	PR5 = 000240	S1.BEL= 000400	T\$KMG= 000000
FE.EXP= 000200	KINAR0= 172340	PR6 = 000300	S1.CTO= 000040	T\$LWC= 000000
FE.EXT= 000001	KINAR5= 172352	PR7 = 000340	S1.CTS= 010000	T\$M11= 000001
FE.EXV= 000004	KINAR6= 172354	PS = 177776	S1.DEC= 002000	T\$RED= 000000
FE.FDT= 002000	KINAR7= 172356	P\$BPR= 000063	S1.DPR= 001000	T\$RNE= 000000
FE.LSI= 000400	KISAR0= 172340	P\$CTL= 000000	S1.DSI= 004000	T\$RPR= 000030
FE.MUP= 000002	KISAR5= 172352	P\$FRS= 000310	S1.ESC= 000004	T\$RST= 000000
FE.MXT= 040000		P\$HIL= 003100	S1.IBF= 100000	T\$RUB= 000000

MATBL MACRO M1200 20-JUN-86 16:15 PAGE 3-3
SYMBOL TABLE

T\$SMC= 000000	UISDR5= 177612	U.ADMA 000056	U.DWN 177776	U.VSER= 000120
T\$SYN= 000000	UISDR6= 177614	U.AFLG 000054	U.RBNS= 000112	U2.AT.= 000020
T\$S11= 000001	UISDR7= 177616	U.ATT 000022	U.RCTC= 000113	U2.CRT= 002000
T\$TRW= 000000	UM.CLI= 000036	U.BPKT= 000044	U.RCTS= 000110	U2.DH1= 100000
T\$UTB= 000000	UM.DSB= 000200	U.BUF 000024	U.RED 000002	U2.DJ1= 040000
T\$UTD= 000170	UM.NBR= 000400	U.CBF = 000032	U.RED2 000034	U2.DZ1= 000100
T\$U58= 000001	UM.QVR= 000001	U.CLI 177772	U.SCB 000020	U2.ESC= 001000
T\$VBF= 000000	US.ABD= 000001	U.CMST= 000126	U.SHST= 000076	U2.HFF= 010000
T\$30P= 000000	US.BSP= 000002	U.CNT 000030	U.SHUN= 000074	U2.HLD= 000040
UBMPR = 170200	US.BSY= 000200	U.COTQ 000030	U.SPC = 000036	U2.LOG= 000400
UC.ALG= 000200	US.FGR= 000040	U.CTCB 000026	U.STS 000005	U2.LWC= 000001
UC.ATT= 000010	US.FRK= 000002	U.CTL 000004	U.ST2 000007	U2.L3S= 000004
UC.KIL= 000004	US.KPF= 000001	U.CTYP 000050	U.SUB = 000036	U2.L8S= 010000
UC.LGH= 000003	US.LAB= 000004	U.CW1 000010	U.TCHP 000042	U2.NEC= 004000
UC.NPR= 000100	US.MDE= 000002	U.CW2 000012	U.TCVP 000043	U2.PRV= 000010
UC.PWF= 000020	US.MDM= 000020	U.CW3 000014	U.TFLK 000040	U2.RMT= 020000
UC.QUE= 000040	US.MNT= 000100	U.CW4 000016	U.TFRQ 000037	U2.R04= 100000
UDSAR0= 177660	US.QFL= 000001	U.CYL = 000104	U.TLPP 000036	U2.SCS= 000004
UDSDR0= 177620	US.PUB= 000004	U.DCS 000000	U.TMTI 000047	U2.SLV= 000200
UD.UMS= 000000	US.PWF= 000010	U.FCDE= 000042	U.TRCK= 000100	U2.VT5= 000002
UD.160= 000004	US.RED= 000002	U.FNUM= 000040	U.TSTA 000026	U2.7CH= 010000
UD.200= 000001	US.SHR= 000001	U.GRP = 000102	U.TTAB 000034	U3.UPC= 020000
UD.556= 000002	US.SPU= 000002	U.HSTI= 000054	U.TTYP 000046	U4.CR = 000100
UD.625= 000005	US.UND= 000010	U.KCSR= 000032	U.TUX 000024	V\$CTR= 000400
UD.800= 000003	US.VV = 000001	U.KCS6= 000034	U.UIC 000044	V\$RSN= 000040
UISAR0= 177640	US.WCK= 000010	U.LHD = 000040	U.UNFL= 000052	\$MADAT 000000R6
UISAR4= 177650	U\$MHI= 000000	U.LUIC 177774	U.UNIT 000006	\$MAEND 000200R6
UISAR5= 177652	U\$MLD= 160000	U.MEDI= 000070	U.UNSZ= 000114	\$MATBL= 000000
UISAR6= 177654	U\$MRN= 170234	U.MLUN= 000050	U.UNTI= 000060	\$MAO 000150R6
UISAR7= 177656	U.ACB 000052	U.MUP 177772	U.UTMC= 000036	.MAO 000040R6
UISDR0= 177600	U.ACP = 000032	U.DTRF= 000124	U.VCS = 000034	.MA1 000106R6
UISDR4= 177610				

. ABS. 177776 000
000200 001

ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 11968 WORDS (47 PAGES)

DYNAMIC MEMORY: 12724 WORDS (48 PAGES)

ELAPSED TIME: 00:00:49

MATBL,MATBL/-SP=LB:C1,1JEXEMC/ML,LB:C11,10JRSXMC,SY:C100,34JMATBL

5086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 16

LOC	OBJ	LINE	SOURCE
01FA	680D		
01FC	F33F		
01FE	7A2E		
0200	0EE7	404	dw 0E70EH,0F687H,0C41CH,0D595H,0A12AH,08CA3H,08236H,09381H
0202	87F6		
0204	1CC4		
0206	95D5		
0208	2AA1		
020A	A3B0		
020C	3882		
020E	B193		
0210	4668	405	dw 06B46H,07ACFH,04854H,059D0H,02D62H,03CEBH,00E70H,01FF9H
0212	CF7A		
0214	5448		
0216	DD59		
0218	622D		
021A	EB3C		
021C	700E		
021E	F91F		
0220	8FF7	406	dw 0F78FH,0E606H,0D49DH,0C514H,0B1A6H,0A022H,09289H,0833DH
0222	06E6		
0224	9DD4		
0226	14C5		
0228	A8B1		
022A	22A0		
022C	B992		
022E	3083		
0230	C77B	407	dw 078C7H,06A4EH,058D5H,0495CH,03DE3H,02C6AH,01EF1H,00F78H
0232	4E6A		
0234	D558		
0236	5C49		
0238	E33D		
023A	6A2C		
023C	F11E		
023E	780F		
		408	
		409 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 17

LOC	OBJ	LINE	SOURCE
		410	-----
		411	public Init_coms
0240		412	Init_coms proc far
		413	;
		414	; FUNCTION : Sets up the data link. Interrupts are set up, buffers cleaned,
		415	; frame numbers set to 0, baud rate to 9600, real time clock
		416	; set up.
		417	;
		418	; INPUTS : None.
		419	;
		420	; CALLS : None.
		421	;
		422	; OUTPUTS : None.
		423	;
		424	; DESTROYS : AX, DX, Flags.
		425	;
		426	;
		427	
		428	
0240	1E	429	push DS
		430	
		431	assume DS: Int_pointers ;First we set up the interrupt
		432	; vectors
0241	880000	433	mov AX, Int_pointers
0244	8ED8	434	mov DS, AX
		435	
0246	B8----	436	mov AX, C_code ;First set up the high part
0249	A33200	437	mov Type_12_seg, AX
024C	A33600	438	mov Type_13_seg, AX
024F	A34E00	439	mov Type_19_seg, AX
		440	
0252	B84A05	441	mov AX, offset Tx_int ;The transmitter interrupt
0255	A33000	442	mov Type_12_offset, AX
0258	B8F704	443	mov AX, offset Rec_int ;The receiver interrupt
025B	A33400	444	mov Type_13_offset, AX
025E	B8FB05	445	mov AX, offset Time_int ;The timer interrupt
0261	A34C00	446	mov Type_19_offset, AX
		447	
		448	
		449	assume DS : C_data
		450	
0264	B8----	451	mov AX, C_data ;Now we want the normal data area
0267	8ED8	452	mov DS, AX

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 15

LOC	OBJ	LINE	SOURCE
		453	
0269	BA52FF	454	mov DX, MCNT0A_reg ;Load the address of the
		455	; baud rate generator
026C	B80500	456	mov AX, Baud_96_a
026F	EF	457	out DX, AX
		458	
0270	BA54FF	459	mov DX, MCNT0B_reg ; and also the other counter
0273	B80500	460	mov AX, Baud_96_b
0276	EF	461	out DX, AX
		462	
0277	BA56FF	463	mov DX, TMROCNTN_reg ;Now the control register
027A	B803C0	464	mov AX, TMR_baud_control ;Set up as baud rate generator
027D	EF	465	out DX, AX
		466	
027E	BA0204	467	mov DX, USART0_control ;Now set up the USART
0281	B00A	468	mov AL, Dummy_cmd
0283	EE	469	out DX, AL ;Ensure that the USART is
0284	EE	470	out DX, AL ; in a predictable mode
0285	EE	471	out DX, AL
0286	B84000	472	mov AX, USART_reset ;Now reset it
0289	EE	473	out DX, AL
028A	90	474	nop ;Give it time
028B	90	475	nop
028C	90	476	nop
028D	90	477	nop
028E	90	478	nop
028F	90	479	nop
0290	90	480	nop
0291	90	481	nop
0292	B0CE	482	mov AL, USART_mode_8 ;Set up for eight bits
0294	EE	483	out DX, AL
0295	B037	484	mov AL, USART_cmd ;Enable the USART
0297	EE	485	out DX, AL
0298	BA0004	486	mov DX, USART0_data ;Now clear any junk in the
		487	; buffers
029B	EC	488	in AL, DX
029C	EC	489	in AL, DX
		490	
029D	BA62FF	491	mov DX, MCNT2A_reg ;Now we set up the clock
02A0	B850C3	492	mov AX, Prescale ;That's the division ratio
02A3	EF	493	out DX, AX
02A4	BA66FF	494	mov DX, TMR2CNTR_reg ;Set up the control register
02A7	B801E0	495	mov AX, TMR_clock_cmd

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 19

LOC	OBJ	LINE	SOURCE
02AA	EF	496	out DX, AX
		497	
02AB	C7062E020000	498	mov Count, 0 ;No timeouts
		499	
02B1	C6069730010	500	mov Status, mask Buf_empty ;The receive buffer is empty
		501	
02B6	C70602000000	502	mov R_length, 0 ;No characters yet received
02BC	C70622020000	503	mov R_state, 0
02C2	C7062802FFFF	504	mov R_crc, Crc_init
02C8	C70604000000	505	mov R_in_ptr, 0
02CE	C70606000000	506	mov R_out_ptr, 0
		507	
		508	;Now set up the interrupt
		509 +1	; controller
02D4	BA32FF	510 +2	mov DX, INT_TMC_reg
02D7	B80700	511 +2	mov AX, INT_TMC_val
02DA	EF	512 +1	out DX, AX
		513 +1	
		514 +1	
02DB	BA38FF	515 +2	mov DX, INT_INT0_reg
02DE	B81800	516 +2	mov AX, INT_INT0_val
02E1	EF	517 +1	out DX, AX
		518 +1	
		519 +1	
02E2	BA3AFF	520 +2	mov DX, INT_INT1_reg
02E5	B81400	521 +2	mov AX, INT_INT1_val
02E8	EF	522 +1	out DX, AX
		523 +1	
		524	
02E9	1F	525	pop DS
02EA	CB	526	ret
		527	
		528	Init_coms endp
		529	
		530 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 20

LOC OBJ

LINE

SOURCE

```

531 ;-----
532 ; Procedure Set_baud
533 ;
534 ; FUNCTION : Set the baud rate of either the data link or the terminal.
535 ;
536 ; INPUTS : Sb_b_sel : Baud rate select :
537 ;           0 - 110 baud.
538 ;           1 - 150 baud.
539 ;           2 - 300 baud.
540 ;           3 - 600 baud.
541 ;           4 - 1200 baud.
542 ;           5 - 2400 baud.
543 ;           6 - 4800 baud.
544 ;           7 - 9600 baud.
545 ;
546 ;           Sb_ch_num : Channel select :
547 ;           0 - Data link
548 ;           1 - Terminal
549 ;
550 ; CALLS : None.
551 ;
552 ; OUTPUTS : None.
553 ;
554 ; DESTROYS : AX, BX, DX, Flags.
555 ;
556 ;
557
558 Set_baud_struct      struc
559
0000 560 Sb_old_BP      dw      ?
0002 561 Sb_return      dd      ?
0006 562 Sb_b_sel      dw      ?
0008 563 Sb_ch_num     dw      ?
564
565 Set_baud_struct      ends
566
567 public              Set_baud
568
02EB 569 Set_baud      proc    far
570
571
02EB C8000000 572                enter    0,0
573

```

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 21

LOC	OBJ	LINE	SOURCE
02EF	8B5E06	574	mov BX, [BP].Sb_b_sel ;Get the baud rate index
02F2	C1E302	575	shl BX, 2 ;Multiply by four to get the correct
			index
02F5	83F820	576	cmp BX, size Baud_tbl ;Is this a valid baud rate?
02F8	7D23	577	jge Sb_exit ;If greater then not
		578	
02FA	8A52FF	579	mov DX, MCNT0A_reg ;Assume channel 0 (terminal)
02FD	8B4608	580	mov AX, [BP].Sb_ch_num ;Get the channel number
0300	3D0000	581	cmp AX, 0 ;Is it zero?
0303	7408	582	je Set_bd ;If yes then fine
0305	3D0100	583	cmp AX, 1 ;Channel 1?
0308	7513	584	jne Sb_exit ;If not then error
030A	8A5AFF	585	mov DX, MCNT1A_reg ;Load up for channel 1
		586	
0300	2E8B4720	587	Set_bd: mov AX, CS:Baud_tbl[EBX] ;Get the first A reg value
0311	EF	588	out DX, AX
		589	
0312	83C302	590	add BX, 2 ;Now to get the B reg value
0315	83C202	591	add DX, 2
0318	2E8B4720	592	mov AX, CS:Baud_tbl[EBX]
031C	EF	593	out DX, AX
		594	
031D	C9	595	Sb_exit: leave
031E	CA0400	596	ret size Set_baud_struct - 6
		597	
		598	Set_baud endp
		599	
		600	
		601	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 22

LCC OBJ

LINE

SOURCE

```

602 ;-----
603 ;Procedure R_frame
604 ;
605 ; FUNCTION : Transfers a received frame to the calling program.
606 ;
607 ; INPUTS :
608 ;     R_error_addr : address of the error flag
609 ;     R_fnum_addr  : address of the frame number
610 ;     R_index      : index into the data table
611 ;     R_data_addr  : address of the data area
612 ;     R_length_addr : address of the length of the frame
613 ;     R_tchar_addr : address of the end character of the frame
614 ;     R_fchar_addr : address of the start character of the frame
615 ;
616 ; CALLS : None.
617 ;
618 ; OUTPUTS : Data as defined above written.
619 ;
620 ; DESTROYS : AX, BX, CX, SI, DI, ES, Flags.
621 ;
622 ;
623
624 R_frame_struct      struc
0000 625 R_old_BP          dw  ?
0002 626 R_return          dd  ?
0006 627 R_error_addr     dd  ?
000A 628 R_fnum_addr      dd  ?
000E 629 R_index         dw  ?
0010 630 R_data_addr      dd  ?
0014 631 R_length_addr    dd  ?
0018 632 R_tchar_addr     dd  ?
001C 633 R_fchar_addr     dd  ?
634
635 R_frame_struct      ends
636
637
638 public             R_frame
639
640
0321 641 R_frame           proc  far
642
643
0321 644               enter  0,0

```

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 23

LOC	OBJ	LINE	SOURCE
0325	1E	645	push DS
0326	88----	646	mov AX, C_data
0329	8ED8	647	mov DS, AX
		648	
		649 +1	
0328	A09000	650 +2	mov AL, R_error ;Get the data
032E	C45E06	651 +2	les BX, [BP]. R_error_addr ;The address of where it goes
0331	268807	652 +1	mov byte ptr ES:[BX], AL ; and store it
		653 +1	
		654 +1	
0334	A10000	655 +2	mov AX, R_f_num ;Get the data
0337	C45E0A	656 +2	les BX, [BP]. R_fnum_addr ;The address of where it goes
033A	268907	657 +1	mov word ptr ES:[BX], AX ; and store it
		658 +1	
		659 +1	
033D	A10200	660 +2	mov AX, R_length ;Get the data
0340	C45E14	661 +2	les BX, [BP]. R_length_addr ;The address of where it goes
0343	268907	662 +1	mov word ptr ES:[BX], AX ; and store it
		663 +1	
		664 +1	
0346	A09100	665 +2	mov AL, R_t_char ;Get the data
0349	C45E18	666 +2	les BX, [BP]. R_tchar_addr ;The address of where it goes
034C	25FF00	667 +1	and AX, 0FFh
034F	268907	668 +1	mov word ptr ES:[BX], AX ; and store it
		669 +1	
		670 +1	
0352	A09200	671 +2	mov AL, R_f_char ;Get the data
0355	C45E1C	672 +2	les BX, [BP]. R_fchar_addr ;The address of where it goes
0358	25FF00	673 +1	and AX, 0FFh
035B	268907	674 +1	mov word ptr ES:[BX], AX ; and store it
		675 +1	
		676	
035E	C47E10	677	les DI, [BP].R_data_addr ;Get the data buffer
		678	; address
0361	037E0E	679	add DI, [BP].R_index
0364	BE0800	680	mov SI, offset R_data ;Get ready to transfer the
		681	; received frame
0367	8B0E0200	682	mov CX, R_length
036B	FC	683	cld ;Set increment
		684	
036C	F3	685	rep movsb ;Do it
036D	A4		
		686	

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 24

LOC	OBJ	LINE	SOURCE
036E	C7062802FFFF	667	mov R_crc, Crc_init
		688	
0374	80289300F7	689	and Status, not mask Error
0379	800E930010	690	or Status, mask Buf_empty ;We can receive again
		691	
037E	C70602000000	692	mov R_length, 0
		693	
		694	
0384	1F	695	R_frame_exit: pop DS
0385	C9	696	leave
0386	CA1A00	697	ret size R_frame_struct - 6
		698	
		699	R_frame endp
		700	
		701 +1	\$EJECT

S086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 25

```

LOC OBJ      LINE      SOURCE
              702      ;-----
              703      ;Procedure T_frame
              704      ;
              705      ; FUNCTION : Transmits a frame to the host system.
              706      ;
              707      ; INPUTS :
              708      ;     T_index      : index into the data table
              709      ;     T_data_addr   : address of the data area
              710      ;     T_fnum_par   : frame number
              711      ;     T_length_par  : frame length
              712      ;     T_tchar_par  : end character
              713      ;     T_fchar_par  : start character
              714      ;
              715      ; CALLS : None.
              716      ;
              717      ; OUTPUTS : None.
              718      ;
              719      ; DESTROYS : AX, BX, CX, SI, DI, ES, Flags.
              720      ;
              721      ;
              722      ;
              723      T_frame_struct  struc
              724      ;
0000          725      T_old_BP      dw      ?
0002          726      T_return     dd      ?
0006          727      T_index      dw      ?
0008          728      T_data_addr   dd      ?
000C          729      T_fnum_par   dw      ?
000E          730      T_length_par dw      ?
0010          731      T_tchar_par  db      ?
0011          732      x            db      ?
0012          733      T_fchar_par  db      ?
0013          734      xx           db      ?
              735      ;
              736      ;
              737      T_frame_struct ends
              738      ;
              739      ;
              740      public      T_frame
              741      ;
0389          742      T_frame      proc    far
              743      ;
              744      ;

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 26

LOC	OBJ	LINE	SOURCE
0387	C8000000	745	enter 0,0
0380	1E	746	push 05
038E	B8----	747	mov AX, C_data
0391	8ED8	748	mov DS, AX
		749	
0393	FB	750	sti
0394	F606930004	751	TX_busy?: test Status, mask TX_busy
0399	75F9	752	jnz TX_busy?
039B	800E930004	753	or Status, mask TX_busy
		754	
03A0	8B460C	755	mov AX, [BP].T_fnum_par ;Get the frame number
03A3	A39400	756	mov T_f_num, AX ;Store it
03A6	8B460E	757	mov AX, [BP].T_length_par ;Get the frame length
03A9	A39600	758	mov T_length, AX
03AC	8BC8	759	mov CX, AX ;Store length for later use
03AE	8A4610	760	mov AL, [BP].T_tchar_par ;Get the terminating character
03B1	A22002	761	mov T_t_char, AL ; and store it
03B4	8A4612	762	mov AL, [BP].T_fchar_par ;Now the header character
03B7	A22102	763	mov T_f_char, AL
		764	
03BA	C45E08	765	les BX, [BP].T_data_addr ;Now get the data address
		766	
03B0	8B7E06	767	mov DI, [BP].T_index ;Initialize the loop index
03C0	BE0000	768	mov SI, 0
03C3	E30B	769	jcxz End_t_frame ;If no data then finished
		770	
03C5	268A01	771	Next_data: mov AL, byte ptr ES:[BX][EDI] ;Get a data byte
03C8	88849800	772	mov T_data[EDI], AL ; and store it
03CC	47	773	inc DI
03CD	46	774	inc SI
03CE	E2F5	775	loop Next_data ;End of data?
		776	
03D0	C70624020000	777	End_t_frame: mov T_state, 0 ;Set the transmitter state
03D6	C70626020000	778	mov Tr_index, 0 ;How many have been sent
03DC	C7062A02FFFF	779	mov T_crc, Crc_init
		780	
03E2	8A28FF	781	mov DX, INT_MSK_reg ;Get ready to enable interrupts
		782	
03E5	ED	783	in AX, DX ;Get the current mask settings
03E6	25EFFF	784	and AX, not mask T0_int ;Enable
03E9	EF	785	out DX, AX
		786	
03EA	1F	787	pop DS

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 27

LOC	OBJ	LINE	SOURCE
03EB	C9	788	leave
03EC	CA0E00	789	ret size T_frame_struct - 6
		790	
		791	
		792	T_frame endp
		793	
		794 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 28

LOC 08J

LINE

SOURCE

```

795 ;-----
796 ;Procedure Set_timer
797 ;
798 ; FUNCTION : Set the timeout value.
799 ;
800 ; INPUTS : Time : The timeout value in 30 Hz ticks.
801 ;
802 ; CALLS : None.
803 ;
804 ; OUTPUTS : None.
805 ;
806 ; DESTROYS : AX, Flags.
807 ;
808 ;
809
---- 810 Set_timer_struct struc
811
0000 812 S_old_BP      dw      ?
0002 813 S_return      dd      ?
0006 814 Time         dw      ?
815
---- 816 Set_timer_struct ends
817
818 public Set_timer
819
03EF 820 Set_timer      proc     far
821
822
03EF C8000000 823             enter     0,0
03F3 1E      824             push     DS
03F4 B8----   R 825             mov      AX, C_data
03F7 8ED8     826             mov      DS, AX
827
828
03F9 884606   829             mov      AX, [BP].Time      ;Get the time value
03FC A32E02   830             mov      Count, AX          ;Store it
831
03FF 1F      832             pop      DS
0400 C9      833             leave
0401 CA0200   834             ret      size Set_timer_struct - 6
835
836 Set_timer      endp
837 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 29

```

LOC OBJ          LINE    SOURCE
                  838      ;-----
                  839      ;Procedure Asm_frame
                  840      ;
                  841      ; FUNCTION : Assembles the received frame from characters stored in the
                  842      ;                circular receive buffer. When a complete frame is received a
                  843      ;                software interrupt is generated. Note that this procedure is
                  844      ;                originally called from the receive interrupt, but then
                  845      ;                continues until the receive buffer is empty.
                  846      ;
                  847      ; INPUTS : None.
                  848      ;
                  849      ; CALLS : None.
                  850      ;
                  851      ; OUTPUTS : None.
                  852      ;
                  853      ; DESTROYS : AX, BX, DX, Flags.
                  854      ;
                  855      ;
                  856
0404              857      Asm_frame      proc      near
                  858
0404 FB           859      Start_asm:      sti
0405 8B1E0600       860                  mov      BX, R_out_ptr
0409 43            861                  inc      BX
040A 81E3FF00      862                  and      BX, 0FFh
040E 8A872001      863                  mov      AL, R_buf[EBX]
0412 32E4          864                  xor      AH, AH
0414 891E0600      865                  mov      R_out_ptr, BX
                  866
0418 50           867                  push     AX
0419 8B1E2802       868                  mov      BX, R_crc           ;Now generate the new crc
041D 32D8          869                  xor      BL, AL
041F 8AC7          870                  mov      AL, BH
0421 32FF          871                  xor      BH, BH
0423 D1E3          872                  shl      BX, 1
0425 2E334740      873                  xor      AX, CS:Crc_tbl[EBX]
0429 A32802        874                  mov      R_crc, AX
                  875
042C 58           876                  pop      AX
                  877
042D 8B1E2202      878                  mov      BX, R_state       ;Get ready to jump via the
                  879                                      ; state table
0431 D1E3          880                  shl      BX, 1           ;Need a word offset

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 30

LOC	OBJ	LINE	SOURCE
0433	FF062202	881	inc R_state ;Assume that we will go
		882	; to the next state
0437	2EFF27	883	jmp CS:R_tableCBXJ ;Do the state jump
		884	
		885	
043A	C70622020100	886	DLE?: mov R_state, i
		887	
0440	C70602000000	888	Rec_dle: mov R_length, 0
0446	3C10	889	cmp AL, DLE ;Did we receive a DLE
0448	7411	890	je Dle_rec
044A	80269300F7	891	Start?: and Status, not mask Error ;If not then carry on waiting
044F	C7062802FFFF	892	mov R_crc, Crc_init
0455	C70622020000	893	mov R_state, 0
045B	E98400	894	Dle_rec: jmp Com_rec_exit
		895	
		896	
		897	
045E	A29200	898	Rec_fchar: mov R_f_char, AL ;Store the received character
		899	
0461	3C02	900	cmp AL, Data_char ;Is it a data char?
0463	7408	901	je F_char_ok ;If so then no error
0465	3C06	902	cmp AL, ACK ;Or maybe ACK?
0467	7404	903	je F_char_ok
0469	3C15	904	cmp AL, NAK ;Last chance!
046B	75CD	905	jne DLE? ;Error, go back and try again
		906	
046D	EB7390	907	F_char_ok: jmp Com_rec_exit
		908	
		909	
		910	
0470	A30000	911	Rec_fnum: mov R_f_num, AX ;Store the frame number
0473	E36090	912	jmp Com_rec_exit
		913	
0476	3C10	914	Rec_data: cmp AL, DLE ;If DLE then maybe end of data
0478	741D	915	je End_rec_data ;Go look for a terminator
		916	
047A	8B1E0200	917	Store_char: mov BX, R_length ;Get the number of characters
		918	; with previous frame yet
047E	81FB8800	919	cmp BX, length R_data ;Is the buffer full?
0482	7F0A	920	jg No_room ;If 6 then full
		921	
0484	8B4708	922	mov R_dataCBXJ, AL ;If we get here then ok
0487	FF060200	923	inc R_length ;Increment the index

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 31

LOC	OBJ	LINE	SOURCE
		924	
048B	EB0690	925	jmp Rec_d_ok
		926	
		927	
048E	800E930008	928	No_room: or Status, mask Error ;Set the error flag
		929	
0493	FF0E2202	930	Rec_d_ok: dec R_state ;We want to stay in this state
		931	
0497	EB4990	932	End_rec_data: jmp Com_rec_exit
		933	
		934	
		935	
		936	
049A	FF0E2202	937	Rec_tchar: dec R_state ;Assume that this is data, and
		938	; set the state accordingly
049E	3C10	939	cmp AL, 0LE ;Is this data?
04A0	7408	940	je Store_char ;If 0LE then yes
		941	
04A2	A29100	942	mov R_t_char, AL ;Else terminating character
04A5	C70622020500	943	mov R_state, 5 ;Get ready for check sum
		944	
04AB	3C04	945	cmp AL, End_char ;Was it an end character?
04AD	7404	946	je T_char_ok
04AF	3C03	947	cmp AL, CONT ;Only other possibility is Cont
04B1	7597	948	jne Start? ;Try Data
04B3	EB2090	949	T_char_ok: jmp Com_rec_exit
		950	
		951	
04B6	813E2802B8F0	952	Rec_bcc: cmp R_crc, CRC_final ;The check sum should be 0
04BC	7405	953	je Cksum_ok
04BE	800E930008	954	or Status, mask Error
04C3	C7062802FFFF	955	Cksum_ok: mov R_crc, Crc_init ;Reset the check sum
04C9	C606900000	956	mov R_error, False ;Assume no errors
04CE	F606930008	957	test Status, mask Error ;Were there errors?
04D3	7405	958	jz Frame_ok ;If Z then not
04D5	C6069000FF	959	mov R_error, True ;Signal so to the block handler
04DA	C70622020000	960	Frame_ok: mov R_state, 0 ;Get ready for the next frame
		961	
04E0	CD64	962	int Frame_rec_int ;Signal that a frame has been
		963	; received
		964	
04E2		965	No_int:
		966	

3066/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 32

LOC	OBJ	LINE	SOURCE
		967	
04E2	FA	968	Com_rec_exit: cli
04E3	8B1E0600	969	mov BX, R_out_ptr
04E7	3B1E0400	970	cmp BX, R_in_ptr
04EB	7403	971	je End_asm
04ED	E914FF	972	jmp Start_asm
04F0	80269300FE	973	End_asm: and Status, not mask RX_busy
04F5	FB	974	sti
04F6	C3	975	ret
		976	
		977	Asm_frame andp
		978	
		979 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 33

```

LOC 09J          LINE    SOURCE
                  980      ;-----
                  981      ;Procedure Rec_interrupt
                  982      ;
                  983      ; FUNCTION : Processes received characters. The character is inserted into a
                  984      ;           circular buffer and a call to Asm_frame is done, unless Asm_frame
                  985      ;           is busy.
                  986      ;
                  987      ; INPUTS : None.
                  988      ;
                  989      ; CALLS : Asm_frame.
                  990      ;
                  991      ; OUTPUTS : None.
                  992      ;
                  993      ; DESTROYS : None.
                  994      ;
                  995      ;
04F7             996
                  997      Rec_int      proc      far
                  998
                  999
04F7 50          1000             push     AX
04F8 53          1001             push     BX
04F9 52          1002             push     DX
04FA 1E          1003             push     DS
                  1004
04FB 88-----   R      1005             mov     AX, C_data      ;Set up the addressing
04FE 8ED8        1006             mov     DS, AX
                  1007
0500 8A0004      1008             mov     DX, USART0_data    ;Get ready to get the data
0503 EC          1009             in      AL, DX
0504 8AE0        1010             mov     AH, AL
0506 8A0204      1011             mov     DX, USART0_control ;Get ready to look for errors
0509 EC          1012             in      AL, DX
050A A838        1013             test    AL, mask USART_error
050C 7406        1014             jz      No_rec_err      ;If Z then no error
050E B037        1015             mov     AL, USART_cmd
0510 EE          1016             out     DX, AL
0511 EB2B90      1017             jmp     R_i_exit
                  1018
0514 8B1E0400    1019      No_rec_err:  mov     BX, R_in_ptr
0518 43          1020             inc     BX
0519 81E3FF00    1021             and     BX, 0FFh
051D 891E0400    1022             mov     R_in_ptr, BX

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 34

LOC	OBJ	LINE	SOURCE
0521	88A72001	1023	mov R_buf[EBX], AH
0525	F606930001	1024	test Status, mask RX_busy
052A	7512	1025	jnz R_i_exit
052C	800E930001	1026	or Status, mask RX_busy
0531	BA22FF	1027	mov DX, INT_EOI_reg ;Get ready to re-enable interrupts
0534	B80080	1028	mov AX, NS_eoi ;Non specific end of interrupt
0537	EF	1029	out DX, AX
		1030	
0538	EBC9FE	1031	call Asm_frame
0538	EB0890	1032	jmp R_exit
		1033	
053E	BA22FF	1034	R_i_exit: mov DX, INT_EOI_reg ;Get ready to re-enable interrupts
0541	B80080	1035	mov AX, NS_eoi ;Non specific end of interrupt
0544	EF	1036	out DX, AX
		1037	
0545	1F	1038	R_exit: pop DS ;Clean up and go
0546	5A	1039	pop DX
0547	58	1040	pop BX
0548	58	1041	pop AX
		1042	
0549	CF	1043	iret
		1044	
		1045	
		1046	Rec_int endp
		1047	
		1048	
		1049	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 35

LOC	OBJ	LINE	SOURCE
		1050	-----
		1051	;Procedure Tx_int
		1052	;
		1053	; FUNCTION : Outputs a character.
		1054	;
		1055	; INPUTS : None.
		1056	;
		1057	; CALLS : None.
		1058	;
		1059	; OUTPUTS : None.
		1060	;
		1061	; DESTROYS : None.
		1062	;
		1063	;
054A		1064	
		1065	Tx_int proc far
		1066	
		1067	
		1068	
054A 50		1069	push AX
054B 53		1070	push BX
054C 52		1071	push CX
054D 1E		1072	push DS
		1073	
054E 88----	R	1074	mov AX, C_data ;Set up the addressing
0551 8ED8		1075	mov DS, AX
		1076	
		1077	
0553 8B1E2402		1078	mov BX, T_state ;Get ready to jump via the
		1079	; state table
0557 01E3		1080	shl BX, 1 ;Need a word offset
0559 FF062402		1081	inc T_state ;Assume that we will go
		1082	; to the next state
055D 2EFF670E		1083	jmp CS:T_table[BX] ;Do the state jump
		1084	
		1085	
0561 8010		1086	Tr_dle: mov AL, DLE ;Set up a dle
0563 EB6F90		1087	jmp Com_tx_exit
		1088	
		1089	
0566 A02102		1090	Tr_fchar: mov AL, T_f_char ;Get the start character
0569 EB6F90		1091	jmp Com_tx_exit
		1092	

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 36

LOC	OBJ	LINE	SOURCE
		1093	
056C	A19400	1094	Tr_fnum: mov AX, T_fnum ;Get the frame number
		1095	
056F	833E960000	1096	cmp T_length, 0 ;Is there data to transmit?
		1097	
0574	7506	1098	jne T_fnum_exit ;If so then just carry on
		1099	
0576	C70624020500	1100	mov T_state, 5 ;If not then skip the data
		1101	; transmission
057C	EB5690	1102	T_fnum_exit: jmp Com_tx_exit
		1103	
		1104	
057F	8B1E2602	1105	Tr_data: mov BX, Tr_index ;Get the data index
0583	8A879800	1106	mov AL, T_data[BX] ;Get the data
0587	FF062602	1107	inc Tr_index
058B	3C10	1108	cmp AL, 0LE ;Do we need to stuff a character
058D	7415	1109	je Char_dle
058F	FF0E2402	1110	dec T_state ;If not then stay in this state
		1111	
0593	FF0E9600	1112	Data_check: dec T_length
0597	833E960000	1113	cmp T_length, 0 ;Last character?
		1114	
059C	7506	1115	jne Still_data
059E	C70624020500	1116	mov T_state, 5 ;End of frame
05A4		1117	Still_data:
05A4	EB2E90	1118	Char_dle: jmp Com_tx_exit
		1119	
		1120	
05A7	C70624020300	1121	Tr_data_dle: mov T_state, 3 ;Assume still data
05AD	8010	1122	mov AL, 0LE ;Load a dle
05AF	E8E2	1123	jmp Data_check ;Go check for more data
		1124	
		1125	
05B1	A02002	1126	Tr_tchar: mov AL, T_t_char ;Load the term character
05B4	EB1E90	1127	jmp Com_tx_exit
		1128	
		1129	
05B7	A12A02	1130	Tr_bcc_lo: mov AX, T_crc
05BA	F7D0	1131	not AX ;We transmit the inverse of the
		1132	; calculated CRC
05BC	A32C02	1133	mov T_crc_save, AX
05BF	EB1390	1134	jmp Com_tx_exit
		1135	

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE.

04/29/86 PAGE 37

```

LDC OBJ          LINE    SOURCE
05C2 8A28FF      1136    Tr_bcc:    mov     DX, INT_MSK_reg    ;Now disable transmit interrupts
05C5 ED          1137                in      AX, DX              ;Get the mask word
05C6 001000      1138                or      AX, mask TO_int    ;Disable
05C9 80269300FB  1139                and     Status, not mask TX_busy
05CE EF          1140                out     DX, AX
05CF A12C02      1141                mov     AX, T_crc_save    ;Load the check sum
05D2 8AC4        1142                mov     AL, AH              ; and get the part that we want
                                1143
                                1144
05D4 50          1145    Com_tx_exit:  push    AX              ;Save the character
                                1146
05D5 8B1E2A02    1147                mov     BX, T_crc              ;Now generate the new crc
05D9 32D8        1148                xor     BL, AL
05DB 8AC7        1149                mov     AL, BH
05DD 32FF        1150                xor     BH, BH
05DF D1E3        1151                shl     BX, 1
05E1 32E4        1152                xor     AH, AH
05E3 2E334740    1153                xor     AX, CS:Crc_tbl[CBX]
05E7 A32A02      1154                mov     T_crc, AX
                                1155
05EA 58          1156                pop     AX
                                1157
05EB 8A0004      1158                mov     DX, USART0_data    ;Get ready to send
05EE EE          1159                out     DX, AL
05EF 8A22FF      1160                mov     DX, INT_EOI_reg    ;Get ready to re-enable interrupts
05F2 B80080      1161                mov     AX, NS_eoi          ;Non specific end of interrupt
05F5 EF          1162                out     DX, AX
                                1163
05F6 1F          1164                pop     DS              ;Clean up and go
05F7 5A          1165                pop     DX
05F8 5B          1166                pop     BX
05F9 5B          1167                pop     AX
                                1168
05FA CF          1169                iret
                                1170
                                1171
05FA CF          1172    Tx_int      endp
                                1173
                                1174
                                1175
05FA CF          1176    +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 38

LDC DBJ

LINE

SOURCE

```

1177 ;-----
1178 ;Procedure Time_int
1179 ;
1180 ; FUNCTION : Timer tick interrupt. If the timer reaches zero then a software
1181 ;           interrupt is generated.
1182 ;
1183 ; INPUTS : None.
1184 ;
1185 ; CALLS : None.
1186 ;
1187 ; OUTPUTS : None.
1188 ;
1189 ; DESTROYS : None.
1190 ;
1191 ;
1192
05FB 1193 Time_int      proc      far
1194
1195
1196
05FB 50 1197          push    AX
05FC 52 1198          push    DX
05FD 1E 1199          push    DS
1200
05FE 88---- R 1201          mov     AX, C_data
0601 8E08 1202          mov     DS, AX
1203
0603 833E2E0200 1204          cmp     Count, 0           ;Is the count finished?
0608 7408 1205          je      End_time_int   ;Do nothing
1206
060A FF0E2E02 1207          dec     Count           ;Count down
060E 7502 1208          jnz     End_time_int   ;If still time then do nothing
1209
0610 CD65 1210          int     Time_out_int    ;If time out then signal so
1211
0612 8A22FF 1212 End_time_int: mov     DX, INT_EOI_reg ;Get ready to re-enable interrupts
0615 B80080 1213          mov     AX, NS_eoi      ;Non specific end of interrupt
0618 EF 1214          out     DX, AX
1215
0619 1F 1216          pop     DS           ;Clean up and go
061A 5A 1217          pop     DX
061B 5B 1218          pop     AX
1219

```

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 39

LOC	OBJ	LINE	SOURCE
0610	CF	1220	iret
		1221	
		1222	
		1223	Time_int endp
		1224	
		1225	
----		1226	C_code ends
		1227	end

ASSEMBLY COMPLETE, NO ERRORS FOUND

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 1

Source File: :F3:ADM8CM.SRC
Object File: :F3:ADM8CM.OBJ
Controls Specified: DEBUG, OPTIMIZE(0).

STMT LINE NESTING

SOURCE TEXT: :F3:ADM8CM.SRC

(*****

MODULE FUNCTION :

This module provides block level data communications.

Written : A.D.McGuffog

19 October 1985.

Language : Intel Pascal-86 V3.0.

Other Software Required : Comms_module.

Restrictions : None.

Module description :

This module provides the block level layer of the data link. Frames (acquired by Comms_module) are assembled by this module into blocks. Once an entire block has been acquired the main program is signaled via a software interrupt. The main program then processes the block and then sends a reply block, also via this module. The block is split into frames, and transmitted. All error control is done by this module, and the type of data link is transparent to the main program.

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 2

STMT LINE NESTING

SOURCE TEXT: IF3:ADMBCN.SRC

```
$target(Fixup has Tape_controller_primitives;
$      exports Init_tape, Select, Erase_tape, Read_block;
$      exports Write_block, Write_with_long_gap, Write_EOF;
$      exports Space, Cntrl_status, Off_line, Rewind_tape;
$      exports Set_RTH_high, Set_RTH_low, Diag;
$      has Comms_module;
$      exports Init_coms, Set_baud, R_frame, T_frame;
$      exports Set_timer)

$compact(-const in code-)

$EJECT
```

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 3

STMT LINE NESTING

SOURCE TEXT: F3:ADMBCM.SRC

```

1  56 0 0      module Block_ID;

                        $include(:"f3:admpdf.src")

2  2 0 0 =1     public Tape_controller_primitives;
3  4 0 0 =1     procedure Init_tape;
4  6 0 0 =1     procedure Select(Unit_sel : integer;
                        T_data : Tape_data_type);
5  9 0 0 =1     procedure Erase_tape;
6  11 0 0 =1    procedure Read_block(var Read_buf : buffer;
                        var N_r_char : integer);
7  14 0 0 =1    procedure Write_block(var Write_buf : buffer;
                        N_w_char : integer);
8  17 0 0 =1    procedure Write_with_long_gap(var Write_buf : buffer;
                        N_w_char : integer);
9  20 0 0 =1    procedure Write_EOF;
10 22 0 0 =1    procedure Space(var Sp_count : integer);
11 24 0 0 =1    procedure Cntrl_status(var Status_return : Status_type);
12 26 0 0 =1    procedure Off_line;
13 28 0 0 =1    procedure Rewind_tape;
14 30 0 0 =1    procedure Set_RTH_high;
15 32 0 0 =1    procedure Set_RTH_low;
16 34 0 0 =1    procedure Diag(var R_buf : buffer;
                        Index : integer;
                        var Parity_error : boolean;
                        var State : integer);
17 39 0 0 =1    public Comms_module;
18 41 0 0 =1    procedure Init_coms;
19 43 0 0 =1    procedure Set_baud(Chan_number : integer;
                        Rate_select : integer);
20 46 0 0 =1    procedure R_frame(var T_char : integer;
                        var F_char : integer;
                        var F_length : integer;
                        var F_data : Block;

```


SERIES-III Pascal-86, V3.0

06/05/86

PAGE 4
BLOCK_ID

STMT	LINE	NESTING	SOURCE TEXT: :F3:ADMPDF.SRC
		=1	F_index : integer;
		=1	var F_num : integer;
		=1	var F_error : boolean);
		=1	
21	54	0 0 =1	procedure T_frame(T_f_char : integer;
		=1	T_t_char : integer;
		=1	T_length : integer;
		=1	T_f_num : integer;
		=1	var T_data : Block;
		=1	T_index : integer);
		=1	
22	61	0 0 =1	procedure Set_timer(Time : integer);
		=1	
		=1	
23	64	0 0 =1	public Block_ID;
		=1	
24	66	0 0 =1	const Frame_rec_int = 100;
25	67	0 0 =1	Time_out_int = 101;
26	68	0 0 =1	Block_rec_int = 102;
		=1	
		=1	
27	71	0 0 =1	procedure Init_block_ID;
		=1	
28	73	0 0 =1	procedure Block_in(var In_block : Input_buf_type);
		=1	
29	75	0 0 =1	procedure Block_out(var Out_block : Output_buf_type;
		=1	Block_length : integer);
		=1	
		=1	
30	90	0 0 =1	public Tape_controller_main;
		=1	
31	82	0 0 =1	const Data_length = 5201;
32	83	0 0 =1	Block_length = 5207;
33	84	0 0 =1	N_drives = 1;
34	85	0 0 =1	Clk_den = 6250;
35	86	0 0 =1	Unit_0_MC = 139;
36	87	0 0 =1	Unit_1_MC = 250;
		=1	
		=1	
37	90	0 0 =1	type Tape_data_type = record
		=1	
37	92	0 1 =1	Stop : integer;
38	93	0 1 =1	Master_clock : integer;
39	94	0 1 =1	Long_gap : integer;
40	95	0 1 =1	Short_gap : integer;
41	96	0 1 =1	Start_norm : integer;
42	97	0 1 =1	Start_long : integer;
43	98	0 1 =1	Start_EOF : integer;
44	99	0 1 =1	Start_read : integer;
		=1	
45	101	0 1 =1	end;
		=1	
		=1	
46	104	0 0 =1	Buffer = packed array [0..Data_length] of char;

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 5
BLOCK_ID

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMPDF.SRC
    =1
47 106 0 0 =1          Block      = packed array [0..Block_length] of char;
    =1
48 108 0 0 =1          Status_type = set of (P_err, CRC_err, Length_err, DMA_err,
    =1                      EOT, EOF, SWL, LGX, S_err, RWD, HWL,
    =1                      xx, Time_err, BOT, EOV, PEOV);
    =1
49 112 0 0 =1          Input_buf_type = record case boolean of
    =1
50 114 0 1 =1              true   : (Opcode      : char;
51 115 0 1 =1              Unit_num  : char;
52 116 0 1 =1              Tape_status_in : Status_type;
53 117 0 1 =1              Count_in   : integer;
54 118 0 1 =1              Write_data  : Buffer);
    =1
55 120 0 1 =1              false  : (Write_block : Block);
    =1
56 122 0 1 =1              end;
    =1
57 125 0 0 =1          Output_buf_type = record case boolean of
    =1
58 127 0 1 =1              true   : (Tape_status_out : Status_type;
59 128 0 1 =1              Count_out : integer;
60 129 0 1 =1              Ret_code  : integer;
61 130 0 1 =1              Read_data  : Buffer);
    =1
62 132 0 1 =1              false  : (Read_block   : Block);
    =1
63 134 0 1 =1              end;
    =1
64 136 0 0 =1          var   Block_diag : boolean;
65 137 0 0 =1          Frame_diag : boolean;
    =1
    =1
66 61 0 0          private Block_ID;

    (* Define the various characters used in the frame protocol. *)

67 66 0 0          const  ACK      = 6q;
68 67 0 0          NAK      = 25q;
69 68 0 0          START    = 2q;
70 69 0 0          CONT     = 3q;
71 70 0 0          END_ch   = 4q;

72 72 0 0          Max_f_len = 64;          (* Frame Length *)
73 73 0 0          Time_out_val = 150;      (* 5 second frame time out *)

74 76 0 0          var T_fnum, R_fnum : integer;
75 77 0 0          R_buf, T_buf      : Block;
76 78 0 0          Start_char       : integer;
77 79 0 0          End_char_in      : integer;

```

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 6
BLOCK_ID

STMT	LINE	NESTING	SOURCE TEXT: :F3:ADMBCM.SRC
78	80	0 0	Frame_len_in : integer;
79	81	0 0	Frame_len_out : integer;
80	82	0 0	F_num_in : integer;
81	83	0 0	Error : boolean;
82	84	0 0	R_block_len : integer;
83	85	0 0	T_block_len : integer;
84	86	0 0	Out_count : integer;

\$EJECT

APPENDIX F

SOFTWARE LISTINGS : TAPE CONTROLLER SOFTWARE

<u>Module</u>	<u>PAGE</u>
Initialization	F--2
Pascal-86 logical record system	F-6
Tape control module	F-46
Frame communications module	F-101
Block communications module	F-140
Main module	F-155

8086/87/88/186 MACRO ASSEMBLER INIT188

16/06/86 PAGE 1

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE INIT188

OBJECT MODULE PLACED IN :F3:ADM118.OBJ

ASSEMBLER INVOKED BY: ASM86.86 :F3:ADM118.ASM

LOC	OBJ	LINE	SOURCE
		1 +1	\$DEBUG
		2 +1	\$GENONLY
		3 +1	\$SYMBOLS
		4 +1	\$TYPE
		5 +1	\$XREF
		6	name Init188
		7	*****
		8	*****
		9	;
		10	;
		11	MODULE FUNCTION :
		12	;
		13	This module is start up code for the 80188 SA-Bus CPU card.
		14	;
		15	*****
		16	Written : A.D.McGuffog 28 October 1985.
		17	;
		18	*****
		19	;
		20	Language : Intel ASM 86.
		21	;
		22	*****
		23	;
		24	Other Software Required : None.
		25	;
		26	*****
		27	;
		28	Hardware required : 80188 CPU card.
		29	;
		30	*****
		31	;
		32	Restrictions : EPROM configuration must be as specified below.
		33	;
		34	*****
		35	;
		36	Module description :
		37	;
		38	This module provides initialization for the 80188 card. The card is set up
		39	to support four EPROMs and 512 K of RAM.
		40	;
		41	*****
		42	*****
		43	;
		44 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

INIT188

16/06/86 PAGE 2

LOC	OBJ	LINE	SOURCE
		45	;
		46	;This module is the initialization routine for an iAPX 188 processor card
		47	;It assumes that the memory consists of
		48	;
		49	;
		50	1 2764 at high mem, addressed by UCMS,
		51	3 2764 at high mem, addressed by MC0 - MC2
		52	;
		53	as well as up to 256 K of ram at low mem
		54	;
		55	;
		56	;
		57	;
		58	; Macro definition for outputing an immediate value to a port.
		59	;
		60	;
		61	;
----		62	Restart segment at 0FFFFh
		63	;
0000		64	org 0
0000 EA0000E0FF		65	jmp far ptr Init
		66	;
----		67	restart ends
		68	;
		69	extrn Main : far
		70	;
----		71	Init_hw segment at 0FF0h
		72	;
		73	assume CS : Init_hw
		74	;
FF00		75	p_ equ 0FF00h
FFA0		76	UMCS_reg equ p_ + 0A0h
FFA2		77	LMCS_reg equ p_ + 0A2h
FFA4		78	PACS_reg equ p_ + 0A4h
FFA6		79	MMCS_reg equ p_ + 0A6h
FFA8		80	MPCS_reg equ p_ + 0A8h
		81	;
FE3F		82	UMCS_val equ 0FE3Fh
3FFB		83	LMCS_val equ 03FFBh
007F		84	PACS_val equ 0007Fh
F9FF		85	MMCS_val equ 0F9FFh
843F		86	MPCS_val equ 0843Fh
		87	;
0000		88	Init proc far
		89	;
		90 +1	;
0000 BAA0FF		91 +2	mov DX, UMCS_reg
0003 B83FFE		92 +2	mov AX, UMCS_val
0006 EF		93 +1	out DX, AX
		94 +1	;
		95 +1	;
0007 BAA2FF		96 +2	mov DX, LMCS_reg
000A B8FB3F		97 +2	mov AX, LMCS_val
000D EF		98 +1	out DX, AX
		99 +1	;

8086/87/88/186 MACRO ASSEMBLER INIT168

16/06/86 PAGE 3

LSC OBJ	LINE	SOURCE
	100 +1	
000E BAA4FF	101 +2	mov DX, PACS_reg
0011 B87F00	102 +2	mov AX, PACS_val
0014 EF	103 +1	out DX, AX
	104 +1	
	105 +1	
0015 BAA6FF	106 +2	mov DX, MMCS_reg
0018 B8FF9	107 +2	mov AX, MMCS_val
001B EF	108 +1	out DX, AX
	109 +1	
	110 +1	
001C BAA8FF	111 +2	mov DX, MPCS_reg
001F B83F84	112 +2	mov AX, MPCS_val
0022 EF	113 +1	out DX, AX
	114 +1	
	115	
	116	
0023 EA0000----	E 117	jmp far ptr Main
	118	
	119	Init endp
	120	
----	121	Init_hw ends
	122	end

8086/87/88/186 MACRO ASSEMBLER INIT188

16/06/86 PAGE 4

XREF SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES, XREFS
??SEG	.. SEGMENT		SIZE=0000H PARA PUBLIC
INIT	.. P FAR	0000H	SIZE=0028H INIT_HW 65 88# 119
INIT_HW	.. SEGMENT		SIZE=0028H PARA ABS 71# 73 121
LMCS_REG	NUMBER	FFA2H	77# 96
LMCS_VAL	NUMBER	3FFBH	83# 97
MAIN	.. L FAR	0000H	EXTRN 69# 117
MMCS_REG	NUMBER	FFA6H	79# 106
MMCS_VAL	NUMBER	F9FFH	85# 107
MPCS_REG	NUMBER	FFA8H	80# 111
MPCS_VAL	NUMBER	843FH	86# 112
P	.. . NUMBER	FF00H	75# 76 77 78 79 80
PACS_REG	NUMBER	FFA4H	78# 101
PACS_VAL	NUMBER	007FH	84# 102
RESTART	.. SEGMENT		SIZE=0005H PARA ABS 62# 67
UMCS_REG	NUMBER	FFA0H	76# 91
UMCS_VAL	NUMBER	FE3FH	82# 92

END OF SYMBOL TABLE LISTING

ASSEMBLY COMPLETE, NO ERRORS FOUND

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 1

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE PASCAL86_LOGICAL_RECORD_SYSTEM
 OBJECT MODULE PLACED IN :F3:ADMP10.OBJ
 ASSEMBLER INVOKED BY: ASM86.86 :F3:ADMP10.ASM

LOC	OBJ	LINE	SOURCE
		1 +1	\$DEBUG
		2 +1	\$GENONLY
		3 +1	\$TYPE
		4 +1	\$PAGELENGTH(48)
		5 +1	\$MOD186
		6	;
		7	name Pascal86_logical_record_system
		8	;
		9	*****
		10	*****
		11	;
		12	;
		13	MODULE FUNCTION :
		14	;
		15	This module is the run time interface for Pascal-86 on the SA-Bus kit using
		16	an 80188 CPU, dynamic RAM and a serial line interface to a terminal device.
		17	*****
		18	;
		19	Written : A.D.McGuffog 7 October 1985.
		20	;
		21	*****
		22	;
		23	Language : Intel ASM 86.
		24	;
		25	*****
		26	;
		27	Other Software Required : Designed to run with Intel Pascal-86 V3
		28	;
		29	*****
		30	;
		31	Hardware required : 80188 CPU card with an 8251A installed as USART1.
		32	Serial link starts at 9600 baud 7 bits , no parity.
		33	;
		34	*****
		35	;
		36	Restrictions : Only two files, INPUT and OUTPUT, supported.
		37	NEW and DISPOSE not supported.
		38	;

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 2

LOC OBJ

LINE

SOURCE

```

39 ;*****
40 ;
41 ;           Module description :
42 ;
43 ;The Pascal-86 run-time interface provides three basic services. These are :
44 ;
45 ;     1. File I/O. All IO operations are in the form of file operations.
46 ;        This module allows only two file, input and output which are the
47 ;        serial in and serial out lines of USART 1 on the 80188 card.
48 ;
49 ;     2. Exception handling. Any exception results in a call to the LRS's
50 ;        exception handler. This prints the Pascal-86 error number,
51 ;        parameters and the address at which the error occurred. Error and
52 ;        parameter number are documented in the Pascal-86 manual chapter 14.
53 ;
54 ;     3. Run-time storage management. Run-time storage is required for two
55 ;        reasons :
56 ;         3.1. File descriptor information : The run-time system uses
57 ;            a 48 byte block of memory for each file opened, in order
58 ;            to store file information. Pascal-86 V3 also requires
59 ;            an extra block although this is not documented.
60 ;         3.2. Use of the Pascal NEW and DISPOSE functions. This is
61 ;            not supported by this LRS.
62 ;
63 ;All of these functions are interfaced via calls to predefined procedures.
64 ; These procedures are documented in the Pascal-86 User's guide Appendix K
65 ; and the Run Time Support for iAPX 86,88 Applications manual.
66 ;
67 ;
68 ;*****
69 ;*****
70 ;
71 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 3

LOC	OBJ	LINE	SOURCE
		72 +1	%include(:F3:ADMDEF.ASM)
		=1 73	;
		=1 74	;
		=1 75	;
		=1 76	
		=1 77	
		=1 78	;
		=1 79	
C MACRO		=1 80	codemacro reset
#		=1 81	db OFAh
#		=1 82	db OEAh
#		=1 83	dw Oh
#		=1 84	dw OFFFh
#		=1 85	endm
		=1 86	
		=1 87	; Now the equates for the on_chip timers :
		=1 88	
FF00		=1 89	Per_base equ OFF00h ;Base address of the 80188
		=1 90	; peripheral control block
FF50		=1 91	CNT0_reg equ Per_base + 050h ;The three count registers :
FF52		=1 92	MCNT0A_reg equ Per_base + 052h
FF54		=1 93	MCNT0B_reg equ Per_base + 054h
FF56		=1 94	THR0CNTR_reg equ Per_base + 056h
FF58		=1 95	CNT1_reg equ Per_base + 058h
FF5A		=1 96	MCNT1A_reg equ Per_base + 05Ah
FF5C		=1 97	MCNT1B_reg equ Per_base + 05Ch
FF5E		=1 98	THR1CNTR_reg equ Per_base + 05Eh
FF60		=1 99	CNT2_reg equ Per_base + 060h
FF62		=1 100	MCNT2A_reg equ Per_base + 062h
FF66		=1 101	THR2CNTR_reg equ Per_base + 066h
		=1 102	
		=1 103	;Now the memory chip select registers
		=1 104	
FFA0		=1 105	UNCS_reg equ Per_base + 0A0h ;Upper select
FFA2		=1 106	LNCS_reg equ Per_base + 0A2h ;Lower select
FFA4		=1 107	PACS_reg equ Per_base + 0A4h ;Peripheral select
FFA6		=1 108	MNCS_reg equ Per_base + 0A6h ;Memory select
FFA8		=1 109	MPCS_reg equ Per_base + 0A8h
		=1 110	
FFCA		=1 111	DMA_CO_reg equ Per_base + 0CAh ;DMA control registers
FFDA		=1 112	DMA_C1_reg equ Per_base + 0DAh
FFC8		=1 113	DMA_T0_reg equ Per_base + 0C8h ;DMA transfer count registers
FFD8		=1 114	DMA_T1_reg equ Per_base + 0D8h

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 4

LOC	OBJ	LINE	SOURCE
FFC6		=1 115	DMA_DPH0_reg equ Per_base + 0C6h ;DMA destination high and low
FFD6		=1 116	DMA_DPH1_reg equ Per_base + 0D6h
FFC4		=1 117	DMA_DPL0_reg equ Per_base + 0C4h
FFD4		=1 118	DMA_DPL1_reg equ Per_base + 0D4h
FFC2		=1 119	DMA_SPH0_reg equ Per_base + 0C2h ;DMA source high and low
FFD2		=1 120	DMA_SPH1_reg equ Per_base + 0D2h
FFC0		=1 121	DMA_SPL0_reg equ Per_base + 0C0h
FFD0		=1 122	DMA_SPL1_reg equ Per_base + 0D0h
		=1 123	
		=1 124	; Now the interrupt control registers :
		=1 125	
FF22		=1 126	INT_EOI_reg equ Per_base + 022h ;End of interrupt register
FF24		=1 127	INT_POL_reg equ Per_base + 024h ;Poll register
FF26		=1 128	INT_PST_reg equ Per_base + 026h ;
FF28		=1 129	INT_MSK_reg equ Per_base + 028h ;Mask register
FF2A		=1 130	INT_PRV_reg equ Per_base + 02Ah ;Priority register
FF2C		=1 131	INT_ISR_reg equ Per_base + 02Ch ;In service register
FF2E		=1 132	INT_REQ_reg equ Per_base + 02Eh ;Request register
FF30		=1 133	INT_STS_reg equ Per_base + 030h ;Status
FF32		=1 134	INT_TMC_reg equ Per_base + 032h ;Timer interrupt
FF34		=1 135	INT_DMA0_reg equ Per_base + 034h ;DMA interrupts
FF36		=1 136	INT_DMA1_reg equ Per_base + 036h
FF38		=1 137	INT_INT0_reg equ Per_base + 038h ;External interrupts
FF3A		=1 138	INT_INT1_reg equ Per_base + 03Ah
FF3C		=1 139	INT_INT2_reg equ Per_base + 03Ch
FF3E		=1 140	INT_INT3_reg equ Per_base + 03Eh
		=1 141	
		=1 142	
FE3F		=1 143	UMCS_val equ 0FE3Fh ;Values for the memory registers
3FFB		=1 144	LMCS_val equ 03FFBh ; 4 by 2764s, 512k ram,
007F		=1 145	PACS_val equ 0007Fh ; 3 wait states + ext. peripheral
F9FF		=1 146	MMCS_val equ 0F9FFh ; select block in IO space,
843F		=1 147	MPCS_val equ 0843Fh ; 3 IO wait states + ext.
		=1 148	
8000		=1 149	NS_eoi equ 1000000000000000b ;Non specific end of interrupt
		=1 150	
0005		=1 151	Maxcount0a equ 5 ;Count values for default 9600
0005		=1 152	Maxcount0b equ 5 ; baud serial lines
0005		=1 153	Maxcount1a equ 5
0005		=1 154	Maxcount1b equ 5
C003		=1 155	TMR_baud_control equ 1100000000000011b
		=1 156	;

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 5

```

LOC OBJ          LINE    SOURCE
                =1 157    ; Now the USART equates :
                =1 158
0400             =1 159    PCS_base      equ    0400h                ;The two USART's on the CPU card
0400             =1 160    USART0_data   equ    PCS_base
0402             =1 161    USART0_control equ    PCS_base + 2
0480             =1 162    USART1_data   equ    PCS_base + 128
0482             =1 163    USART1_control equ    PCS_base + 130
00CA            =1 164    USART_mode_7   equ    11001010b        ; 2 stop bits, no parity, 7 bits
00CE            =1 165    USART_mode_8   equ    11001110b        ; 2 stop bits, no parity, 8 bits
0037            =1 166    USART_cmd      equ    00110111b        ; Enable all
0040            =1 167    USART_reset    equ    01000000b
000A            =1 168    Dummy_cmd      equ    00001010b
0002            =1 169    Rec_mask       equ    02h              ; Look for RE ready
0001            =1 170    Tr_mask        equ    01h              ; Look for TX ready
000D            =1 171    CR             equ    0Dh
000A            =1 172    LF             equ    0Ah
                =1 173
                =1 174    ;Bit defs for the bits in the interrupt mask register
                =1 175
#               =1 176    Int_bits       record R1_int:1,T1_int:1,RO_int:1,TO_int:1,
                =1 177    &             DMA_1_int:1,DMA_0_int:1,Int_bits_dummy:1,TMR_int:1
                =1 178
                =1 179    ;Bit defs for the 8251 USART status register
                =1 180
#               =1 181    USART_bits     record DSR:1,SYN_det:1,USART_error:3,TX_empty:1,RX_rdy:1,
                =1 182    &             TX_rdy:1
                =1 183
0008            =1 184    BS             equ    08h
007F            =1 185    DEL            equ    07Fh
                =1 186
                =1 187    ; *** Pascal-86 status returns ***
                =1 188
0000            =1 189    E_ok           equ    0h              ; Status ok for the LRS
0002            =1 190    E_mem         equ    02h              ; Too little memory
15FE            =1 191    E_eor         equ    15FEh
                =1 192
                =1 193    ; *** Internal status returns ***
                =1 194
0000            =1 195    Normal_term   equ    0
00FF            =1 196    Buffer_empty   equ    0FFh
                =1 197
#               =1 198    Status_bits    record Tqn1_used:1, Tqn3_used:1
                =1 199

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 6

```

LOC OBJ          LINE    SOURCE
                200
                201
                202
                203      ;
                204      ;
                205      ;
                206      ;Define a stack segment.
                207      ;
-----          208      stack segment          STACK          'STACK'
                209
                210
                211
0000 (8          212                      dw      8          dup      (?)
      ???
      )
                213
-----          214      stack ends
                215
                216
                217      ; Now a segment for the address table :
                218      ; This is used by Pascal-86 fir indirect calls to file IO routines.
                219
                220
-----          221      Tables          segment          'CODE'
                222
0000 1900----- R    223      Addr_tbl      dd      Open
0004 5F00----- R    224                      dd      Close
0008 6500----- R    225                      dd      Read
000C AE00----- R    226                      dd      Write
0010 CA00----- R    227                      dd      Seek
0014 D000----- R    228                      dd      Skip
0018 F100----- R    229                      dd      E_o_r
001C 0101----- R    230                      dd      Rewind
0020 0701----- R    231                      dd      Backspace
0024 0001----- R    232                      dd      E_o_f
0028 1301----- R    233                      dd      File_info
002C ????????? 234                      dd      ?
                235
                236
0030 01000000   237      File_data      dd      1
0034 01000000   238                      dd      1
0038 ????????? 239                      dd      ?
                240

```

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 7

LOC	OBJ	LINE	SOURCE
		241	
		242	; Pascal-86 termination and exception notification messages
		243	
003C	0D	244	Bad_exit db CR, LF, 'Pascal-86 Fatal Error', CR, LF
003D	0A		
003E	50617363616C2D		
	38362046617461		
	6C204572726F72		
0053	0D		
0054	0A		
0055	0D	245	Good_exit db CR, LF, 'Pascal-86 End', CR, LF
0056	0A		
0057	50617363616C2D		
	383620456E64		
0064	0D		
0065	0A		
		246	
0066	0D	247	Err_msg_1 db CR, LF, 'Pascal-86 Exception Code : '
0067	0A		
0068	50617363616C2D		
	38362045786365		
	7074696F6E2020		
	20202020202020		
	436F6465203A20		
0082	48	248	Err_msg_2 db 'H', CR, LF, ' Parameter number : '
008C	0D		
008D	0A		
008E	20202020202020		
	20202020202020		
	20205061726160		
	65746572206E75		
	6D626572203A20		
0081	48	249	Err_msg_3 db 'H', CR, LF, ' Numeric Exception Code : '
0082	0D		
0083	0A		
0084	20202020202020		
	2020204E756D65		
	72696320457863		
	657074696F6E20		
	436F6465203A20		
0007	48	250	Err_msg_4 db 'H', CR, LF, ' At or Near Location : '
0008	0D		
0009	0A		

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 8

```

LOC DBJ          LINE    SOURCE

000A 202020202020
      20202020202041
      74206F72204E65
      6172204C6F6361
      74696F6E203A20
00F0 48          251      Err_msg_5      db      'H', CR, LF
00FE 0D
00FF 0A

-----          252
          253      Tables          ends
          254
          255
          256      ; Local data space
          257
-----          258      IO_data          segment      'DATA'
          259
0000 7777        260      Local_index      dw      ?          ; index into input buffer
0002 (132        261      Local_buffer     db      132      dup      (?)      ; local character input buffer
      ??
      )
0086 ??          262      Status          db      ?          ; status information storage
0087 ?????????  263      Exception_addr dd      ?          ; storage for the address of the
          264                      ; exception handler
          265
-----          266      IO_data          ends
          267
          268
          269
          270      ; Segments for the file descriptors required by Pascal-86:
          271
          272
          273
-----          274      Tqn1          segment      'DATA'
          275
0000 (48          276                      db      48      dup      (?)
      ??
      )
          277
-----          278      Tqn1          ends
          279
          280
-----          281      Tqn2          segment      'DATA'
          282

```


8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/27/86 PAGE 9

```
LDC GBJ          LINE  SOURCE
0000 (48         283      db      48      dup      (?)
  ??
  )
-----          284
285      Tqn2      ends
286
-----          287      Tqn3      segment      'DATA'
288
0000 (48         289      db      48      dup      (?)
  ??
  )
-----          290
291      Tqn3      ends
292
293
294 +1 $EJECT
```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 10

LOC	OBJ	LINE	SOURCE
		295	
----		296	ID_code segment 'CODE'
		297	
		298	assume CS : ID_code
		299	
		300	public TQDEVICE
		301	
		302	;
		303	;
0000		304	TQDEVICE proc far
		305	;
		306	; FUNCTION : Inserts the address of the driver table into the table pointer.
		307	;
		308	; INPUTS : Tablepointer at [BP + 6].
		309	;
		310	; CALLS : None.
		311	;
		312	; OUTPUTS : Address of the table in the location pointed to by tablepointer.
		313	; Status in AX.
		314	;
		315	; DESTROYS : ES, DI, Flags.
		316	;
		317	;
		318	
		319	
0000 C8000000		320	enter 0,0
0004 C47E06		321	les DI, dword ptr [BP + 6] ; load the pointer
0007 26C7050000		322	mov word ptr ES:[DI], offset Addr_tbl
000C 26C74502----	R	323	mov word ptr ES:[DI + 2], seg Addr_tbl
0012 B80000		324	mov AX, E_ok ; status is ok
0015 C9		325	leave ; clean up
0016 CA0A00		326	ret 10 ; and go
		327	
		328	
		329	TQDEVICE endp
		330	
		331	
		332	
		333 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 11

LOC	OBJ	LINE	SOURCE
		334	-----
		335	
0019		336	Open proc far
		337	;
		338	; FUNCTION : This procedure opens the I/O files by initializing the serial
		339	; port. Note that this is done regardless of file name or type.
		340	;
		341	; INPUTS : None (used).
		342	;
		343	; CALLS : None.
		344	;
		345	; OUTPUTS : None (serial port 1 initialized).
		346	;
		347	; DESTROYS : AX, BX, DX, SI, DI, Flags.
		348	;
		349	;
		350	
		351	
		352	assume DS : IO_data
		353	
0019 1E		354	push DS
001A B8----	R	355	mov AX, IO_data
0010 8ED8		356	mov DS, AX
		357	
		358 +1	
001F 8A5AFF		359 +2	mov DX, MCNT1A_reg
0022 880500		360 +2	mov AX, Maxcount1a
0025 EF		361 +1	out DX, AX
		362 +1	
		363 +1	
0026 8A5CFF		364 +2	mov DX, MCNT1B_reg
0029 880500		365 +2	mov AX, Maxcount1b
002C EF		366 +1	out DX, AX
		367 +1	
		368 +1	
002D 8A5EFF		369 +2	mov DX, TMR1CNTR_reg
0030 8803C0		370 +2	mov AX, Tmr_baud_control
0033 EF		371 +1	out DX, AX
		372 +1	
		373	
0034 8A8204		374	mov DX, USART1_control
0037 800A		375	mov AL, Dummy_cmd ;Do NOT enter sync mode at all costs!
0039 EE		376	out DX, AL ;Dummy stores to get USART ready

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 12

LOC	OBJ	LINE	SOURCE
003A	EE	377	out DX, AL
003B	EE	378	out DX, AL
003C	884000	379	mov AX, USART_reset ;Reset the USART
003F	EE	380	out DX, AL
0040	90	381	nop ;Wait out the required reset time
0041	90	382	nop
0042	90	383	nop
0043	90	384	nop
0044	90	385	nop
0045	90	386	nop
0046	90	387	nop
0047	90	388	nop
0048	80CA	389	mov AL, USART_mode_7 ;Set it up
004A	EE	390	out DX, AL
004B	8037	391	mov AL, USART_cmd ;Enable it
004D	EE	392	out DX, AL
004E	8A8004	393	mov DX, USART1_data
0051	EC	394	in AL, DX ;Get rid of the junk
0052	EC	395	in AL, DX
		396	
		397	
		398	
0053	C6060200FF	399	mov Local_buffer[0], Buffer_empty ; flush the buffer
0058	880000	400	mov AX, E_ok ; set the status
005B	1F	401	pop DS
005C	CA0C00	402	ret 12 ; and go
		403	
		404	
		405	
		406	Open endp
		407	
		408	
		409	assume DS : nothing
		410	
		411	
		412	
		413	
		414	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 13

```

LOC OBJ      LINE      SOURCE
              415      ;-----
              416
005F          417      Close      proc      far
              418      ;
              419      ; FUNCTION : Closes a file (in this case do nothing).
              420      ;
              421      ; INPUTS : None (used).
              422      ;
              423      ; CALLS : None.
              424      ;
              425      ; OUTPUTS : Status in AX.
              426      ;
              427      ; DESTROYS : Flags.
              428      ;
              429      ;
              430
005F 380000   431          mov      AX, E_ok      ; set up for ok return
0062 CA0400   432          ret      4            ; and go
              433
              434
              435      Close      endp
              436
              437
              438
              439
              440
              441 +1 $EJECT

```


8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 14

```

LOC 08J      LINE    SOURCE
              442      ;-----
              443
0065          444      Read      proc      far
              445      ;
              446      ; FUNCTION : Reads a specified number of characters from a file (in this
              447      ;           case always from con1).
              448      ;
              449      ; INPUTS : Number of characters to be read at [BP + 10].
              450      ;           Start address of the buffer into which they are to be read at [SP + 12].
              451      ;           Address to which the number of characters actually transferred must
              452      ;           be written at [BP + 6].
              453      ;
              454      ; CALLS : Get_line.
              455      ;
              456      ; OUTPUTS : Status in AX.
              457      ;           Characters in the specified buffer space.
              458      ;
              459      ; DESTROYS : AX, BX, DX, SI, DI, ES, Flags.
              460      ;
              461      ;
              462
              463      assume      DS : IO_data
              464
0065 C8000000      465      enter      0, 0
0069 1E            466      push      DS
006A 88----        R      467      mov      AX, IO_data
006D 8ED8          468      mov      DS, AX          ; set up for access to
              469      ; local data
006F 803E0200FF    470      cmp      Local_buffer[0], Buffer_empty ; get data?
0074 7503          471      jne      Data_in_buffer ; no.....
0076 E82002        472      call     Get_line
0079 8B4E0A        473      Data_in_buffer: mov     CX, word ptr [BP + 10] ; get the count value
007C C47E0C        474      les      DI, dword ptr [BP + 12] ; and the address of the
              475      ; buffer
007F 8B1E0000      476      mov      BX, local_index ; how far into the line
              477      ; we are
0083 8E0000        478      mov      SI, 0 ; and zero the index for
              479      ; this read operation
0086 FC            480      cld ; set auto increment
0087 8A4002        481      mov      AL, Local_buffer[BX][ESI] ; next character
008A 3C00          482      cmp      AL, CR ; is this the end of line?
008C 7409          483      je      End_of_line
008E AA            484      stosb ; store into the output buffer

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 15

LOC	OBJ	LINE	SOURCE
008F	46	485	inc SI ; next character
0090	E2E7	486	loop Data_in_buffer ; until finished
0092	880000	487	mov AX, E_ok ; if we got here then all
		488	; characters requested were
		489	; transferred
0095	EB08	490	jmp short Read_exit
0097	38FE15	491	End_of_line : mov AX, E_eor
009A	C6060200FF	492	mov Local_buffer[0], Buffer_empty
		493	
009F	C47E06	494	Read_exit : les DI, dword ptr CBP + 6] ; the address for the number of
		495	; characters actually transferred
00A2	268935	496	mov word ptr ES:[DI], SI ; transfer the count
00A5	01360000	497	add Local_index, SI ; adjust the stored record
		498	; position
00A9	1F	499	pop DS ; clean up
00AA	C9	500	leave
00AB	CA0C00	501	ret 12 ; and go
		502	
		503	
		504	Read endp
		505	
		506	
		507	
		508	assume DS : nothing
		509	
		510	
		511	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 16

```

LOC OBJ          LINE    SOURCE
                    512    ;-----
                    513    ;
00AE             514    Write      proc      far
                    515    ;
                    516    ; FUNCTION : Writes to a file ( in this case always to con1 ).
                    517    ;
                    518    ; INPUTS : Number of characters to be sent in [BP + 6].
                    519    ;         Address of the character buffer in [BP + 8].
                    520    ;
                    521    ; CALLS : Conout.
                    522    ;
                    523    ; OUTPUTS : Status in AX (and the characters to the serial line).
                    524    ;
                    525    ; DESTROYS : BX, DX, CX, SI, Flags.
                    526    ;
                    527    ;
                    528    ;
00AE C8000000     529            enter    0,0
00B2 1E           530            push    DS                ; save DS
00B3 8B4E06       531            mov     CX, word ptr [BP + 6]    ; the number of characters
00B6 C57608       532            lds     SI, dword ptr [BP + 8]    ; the address of the buffer
00B9 E307         533            jcxz    End_write                ; just in case of 0 length
                    534    ;
00BB FC           535            cld                        ; set auto increment
00BC AC           536    String_write : lodsb                ; get the next element of the string
00BD E83C02       537            call    Conout                ; output it
00C0 E2FA         538            loop   String_write                ; until no more characters
                    539    ;
00C2 1F           540    End_write : pop     DS                ; restore DS
00C3 880000       541            mov     AX, E_ok                ; set up for ok return
00C6 C9           542            leave                   ; clean up
00C7 CA0800       543            ret     8                        ; and go
                    544    ;
                    545    ;
                    546    Write      endp
                    547    ;
                    548    ;
                    549    ;
                    550    ;
551 +1 $EJECT

```


8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 17

LOC	OBJ	LINE	SOURCE
		552	;------
		553	
00CA		554	Seek proc far
		555	;
		556	; FUNCTION : Sets the file pointer (this function is not used by Pascal-86).
		557	;
		558	; INPUTS : None (used).
		559	;
		560	; CALLS : None.
		561	;
		562	; OUTPUTS : Status in AX.
		563	;
		564	; DESTROYS : Flags.
		565	;
		566	;
		567	
00CA 380000		568	mov AX, E_ok ; set up for ok return
00CD CA0800		569	ret 8 ; and go
		570	
		571	
		572	Seek endp
		573	
		574	
		575	
		576	
		577	
		578 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 18

```

LOC OBJ          LINE    SOURCE
                    579      ;-----
                    580
0000              581      Skip          proc    far
                    582      ;
                    583      ; FUNCTION : Skips to the next record in a file (in this
                    584      ;             case always from con1).
                    585      ;
                    586      ; INPUTS : None.
                    587      ;
                    588      ; CALLS : Get_line.
                    589      ;
                    590      ; OUTPUTS : Status in AX.
                    591      ;
                    592      ; DESTROYS : AX, BX, DX, SI, DI, ES, Flags.
                    593      ;
                    594      ;
                    595      assume      DS : IO_data
                    596
0000 C3000000      597                      enter    0, 0
0004 1E            598                      push     DS
0005 B8-----      599                      mov      AX, IO_data
0008 8ED8           600                      mov      DS, AX
                    601                      ; set up for access to
                    602                      ; local data
                    603
000A 803E0200FF     603                      cmp      Local_buffer[0], Buffer_empty
000F 7503           604                      jne      Skip_exit
                    605                      ; if not empty then empty it
00E1 E8B501         605                      call     Get_line
                    606                      ; if empty then first get
                    607                      ; a line before skipping it
00E4 C6060200FF     607      Skip_exit : mov      Local_buffer[0], Buffer_empty ; flush the buffer
00E9 B80000         608                      mov      AX, E_ok
                    609
00EC 1F            610                      pop      DS
                    611                      ; clean up
00ED C9            611                      leave
                    612
00EE CA0200         612                      ret      2
                    613                      ; and go
                    614
                    614      Skip          endp
                    615
                    616      assume      DS : nothing
                    617
                    618 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 19

```

00F1      E_o_r          proc    far
        ;-----
619       ;
620       ;
621       ; FUNCTION : Writes a record and sequence to a file (in this case a CRLF
622       ;             to conf).
623       ;
624       ; INPUTS : None (used).
625       ;
626       ; CALLS : Conout.
627       ;
628       ; OUTPUTS : Status in AX (and characters to the serial line).
629       ;
630       ; DESTROYS : DX, BX, Flags.
631       ;
632       ;
633       ;
634       ;
635       ;
636       mov     AL, CR
00F1 B00D      call     Conout
00F3 E80602     mov     AL, LF
00F6 B00A      call     Conout
00F8 E80102     mov     AX, E_ok           ; set up for ok return
00FB B80000     ret     2                 ; and go
00FE CA0200
643
644
645     E_o_r          endp
646
647
648
649
650
651
652
653 +1 $EJECT
```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 20

LOC	OBJ	LINE	SOURCE
		654	;-----
		655	
0101		656	Rewind proc far
		657	;
		658	; FUNCTION : Rewinds a file (in this case do nothing).
		659	;
		660	; INPUTS : None (used).
		661	;
		662	; CALLS : None.
		663	;
		664	; OUTPUTS : Status in AX.
		665	;
		666	; DESTROYS : Flags.
		667	;
		668	;
		669	
0101	880000	670	mov AX, E_ok ; set up for ok return
0104	CA0400	671	ret 4 ; and go
		672	
		673	
		674	Rewind endp
		675	
		676	
		677	
		678	
		679	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 21

LOC OBJ

LINE

SOURCE

```
680 ;-----
681 ;
0107 682 Backspace      proc      far
683 ;
684 ; FUNCTION : Backspaces a file (in this case do nothing).
685 ;
686 ; INPUTS : None (used).
687 ;
688 ; CALLS : None.
689 ;
690 ; OUTPUTS : Status in AX.
691 ;
692 ; DESTROYS : Flags.
693 ;
694 ;
695
0107 696                mov     AX, E_ok      ; set up for ok return
010A 697                ret      2          ; and go
698
699
700 Backspace      endp
701
702
703
704
705
706 +1 $EJECT
```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 22

LOC	OBJ	LINE	SOURCE
		707	;-----
		708	
0100		709	E_o_f proc far
		710	;
		711	; FUNCTION : Ends a file (in this case do nothing).
		712	;
		713	; INPUTS : None (used).
		714	;
		715	; CALLS : None.
		716	;
		717	; OUTPUTS : Status in AX.
		718	;
		719	; DESTROYS : Flags.
		720	;
		721	;
		722	
0100	B80000	723	mov AX, E_ok ; set up for ok return
0110	CA0200	724	ret 2 ; and go
		725	
		726	
		727	E_o_f endp
		728	
		729	
		730	+1 \$EJECT

3086/87/88/196 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 23

```

L3C 3BJ          LINE      SOURCE
                    731      ;-----
                    732
0113             733      File_info      proc      far
                    734      ;
                    735      ; FUNCTION : Inserts the address of a table of file information into the
                    736      ;               location pointed to by [BP + 2].
                    737      ;
                    738      ; INPUTS : Tablepointer at [BP + 2].
                    739      ;
                    740      ; CALLS : None.
                    741      ;
                    742      ; OUTPUTS : Address of the table in the location pointed to by tablepointer.
                    743      ;               Status in AX.
                    744      ;
                    745      ; DESTROYS : ES, DI, Flags.
                    746      ;
                    747      ;
                    748      assume DS:Tables
                    749
                    750
0113 C8000000     751          enter    0,0
0117 1E          752          push     DS
0118 88-----    R  753          mov     AX, Tables
0118 8ED8         754          mov     DS, AX
0110 C47E02      755          les     DI, dword ptr [BP + 2] ; load the pointer
0120 26C7053000  756          mov     word ptr ES:[DI], offset File_data
0125 26C74502---- R  757          mov     word ptr ES:[DI + 2], seg File_data
0128 B80000      758          mov     AX, E_ok ; status is ok
012E 1F          759          pop     DS
012F C9          760          leave    ; clean up
0130 CA0600      761          ret     6 ; and go
                    762
                    763
                    764      File_info      endp
                    765
                    766      public      TOEXIT
                    767
                    768
                    769 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 24

```

LOC 03J          LINE    SOURCE
                  770      ;-----
                  771
0133             772      TOEXIT      proc      far
                  773      ;
                  774      ; FUNCTION : This is the PASCAL86 exit routine. In this case a jump to the
                  775      ;           reset vector is done.
                  776      ;
                  777      ; INPUTS : None (used).
                  778      ;
                  779      ; CALLS : Conout.
                  780      ;
                  781      ; OUTPUTS : None.
                  782      ;
                  783      ; DESTROYS : Nothing.
                  784      ;
                  785      ;
                  786
                  787
                  788      assume      DS :      Tables
                  789
0133 C8000000      790      enter      0, 0
0137 B8----      R      791      mov      AX, Tables
013A 8ED8          792      mov      DS, AX      ; Don't bother to save DS
013C FC           793      cld      ; Auto increment.
                  794
013D 837E0600      795      cmp      word ptr [BP + 6], Normal_term
0141 7412          796      je      Normal
                  797 +1
0143 B91900        798 +2      mov      CX, length Bad_exit      ; length of the string
0146 BE3C00        799 +2      mov      SI, offset Bad_exit      ; offset of the string
0149 AC           800 +2      ST_LB_00 : lods      Bad_exit
014A EBAF01        801 +1      call     Conout      ; output the character
014D E2FA          802 +2      loop     ST_LB_00      ; until finished
                  803 +1
014F FAEA0000FFFF  804      reset
                  805
0155             806 +1      Normal :
0155 B91100        807 +2      mov      CX, length Good_exit      ; length of the string
0158 BE5500        808 +2      mov      SI, offset Good_exit      ; offset of the string
015B AC           809 +2      ST_LB_01 : lods      Good_exit
015C E89D01        810 +1      call     Conout      ; output the character
015F E2FA          811 +2      loop     ST_LB_01      ; until finished
                  812 +1

```


8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 25

LOC DBJ

LINE SOURCE

```
0161 FAE40000FFFF      813
                        814      reset      ; disable interrupts and go to the reset vector
                        815
                        816
                        817      TQEXIT      endp
                        818
                        819
                        820      assume      OS :      nothing
                        821
                        822      public      TQALLOCATE
                        823
                        824
                        825 +1 $EJECT
```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 26

LOC	88J	LINE	SOURCE
		826	;-----
		827	
0167		828	TQALLOCATE proc far
		829	;
		830	; FUNCTION : This is the PASCAL86 memory allocation procedure. This procedure
		831	; always returns a no memory available status word.
		832	;
		833	; INPUTS : Address to which the status must be written in [BP + 6].
		834	;
		835	; CALLS : None.
		836	;
		837	; OUTPUTS : Status to [BP + 6].
		838	;
		839	; DESTROYS : ES, DI, Flags.
		840	;
		841	;
		842	assume DS: IO_data
		843	
0167 C8000000		844	enter 0, 0
016B 1E		845	push ds
016C B8----	R	846	mov AX, IO_data
016F 8ED8		847	mov DS, AX
0171 C47E06		848	les DI, dword ptr [BP + 6] ; load the address of the
		849	; status word
0174 26C7050200		850	mov word ptr ES:[DI], E_aem ; assume no memory
0179 F606860001		851	test Status, mask Tqn3_used ; is there space?
017E 7508		852	jnz Alloc_end
0180 26C7050000		853	mov word ptr ES:[DI], E_ok ; ok
0185 38----	R	854	mov AX, Tqn3
0188 1F		855	Alloc_end: pop DS
0189 C9		856	leave ; clean up
018A CA0600		857	ret 6
		858	
		859	TQALLOCATE endp
		860	
		861	public TQFREE
		862	
		863 +1	\$EJECT

3086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 27

```

000 000 LINE SOURCE
0180 00000000 364 ; -----
0181 00000000 365 ;
0182 00000000 366 T@FREE proc far
0183 00000000 367 ;
0184 00000000 368 ; FUNCTION : This is the PASCAL86 memory return procedure. In this case
0185 00000000 369 ; nothing is done.
0186 00000000 370 ;
0187 00000000 371 ; INPUTS : Address to which the status word must be written.
0188 00000000 372 ;
0189 00000000 373 ; CALLS : None.
0190 00000000 374 ;
0191 00000000 375 ; OUTPUTS : Status in [BP + 6].
0192 00000000 376 ;
0193 00000000 377 ; DESTROYS : ES, DI, Flags.
0194 00000000 378 ;
0195 00000000 379 ;
0196 00000000 380
0197 00000000 381 enter 0, 0
0198 00000000 382 les DI, dword ptr [BP + 6] ; load the address of the
0199 00000000 383 ; status word
0200 00000000 384 mov word ptr ES:[DI], E_ok ;
0201 00000000 385 leave ; clean up
0202 00000000 386 ret 6
0203 00000000 387
0204 00000000 388
0205 00000000 389 T@FREE endp
0206 00000000 390
0207 00000000 391
0208 00000000 392
0209 00000000 393 public T@FILE_DESCRIPTOR
0210 00000000 394
0211 00000000 395
0212 00000000 396 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/08/91 PAGE 22

LOC	OBJ	LINE	SOURCE
		897	;------
		898	
0190		899	TQFILEDESCRIPTOR proc far
		900	;
		901	; FUNCTION : Provides the LRS with a 48 byte file descriptor area when
		902	; called. Note that this routine can provide only two such areas.
		903	;
		904	; INPUTS : Address to which the segment number of the data area must be
		905	; written in [BP + 6].
		906	;
		907	; CALLS : None.
		908	;
		909	; OUTPUTS : Status in AX.
		910	;
		911	; DESTROYS : AX, DI, ES, Flags.
		912	;
		913	;
		914	
		915	assume DS : IO_data
		916	
0190	C8000000	917	enter 0, 0
01A1	1E	918	push DS
01A2	B8----	919	mov AX, IO_data
01A5	8ED8	920	mov DS, AX ; set up for access to
		921	; local data
		922	
01A7	C47E06	923	les DI, dword ptr [BP + 6] ; the address of the pointer
01AA	F606860002	924	test Status, mask Tqn1_used ; should we assign Tqn1 or 2
01AF	750C	925	jnz Use_Tqn2
01B1	26C705----	926	mov word ptr ES:[DI], Tqn1 ; supply Tqn1
01B6	800E860002	927	or Status, mask Tqn1_used ; Tqn1 used.
01BB	EB05	928	jmp short TQFILE_exit
01BD	26C705----	929	Use_Tqn2 : mov word ptr ES:[DI], Tqn2 ; supply Tqn2
01C2	B80000	930	TQFILE_exit : mov AX, E_ok ; exit ok
		931	
01C5	1F	932	pop DS ; clean up
01C6	C9	933	leave
01C7	CA0400	934	ret 4 ; and go
		935	
		936	TQFILEDESCRIPTOR endp
		937	
		938	public TQINITIALIZE
		939	

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 29

LOC OBJ

LINE SOURCE

940 +1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 30

LOC	OBJ	LINE	SOURCE
		941	;
		942	;
01CA		943	TQINITIALIZE proc far
		944	;
		945	; FUNCTION : Initializes the LRS data areas, as well as the exception handlers.
		946	;
		947	; INPUTS : None (used).
		948	;
		949	; CALLS : TQSETERH.
		950	;
		951	; OUTPUTS : Status in AX.
		952	;
		953	; DESTROYS : Flags.
		954	;
		955	;
01CA 1E		956	push DS
		957	
		958	assume DS : IO_data
		959	
01CB B8----	R	960	mov AX, IO_data
01CE 8ED8		961	mov DS, AX ; set up for access to
		962	; local data
		963	
01D0 80268600FC		964	and Status, not mask Tqn1_used and not mask Tqn3_used
		965	; no file descriptor used yet
01D5 68----90	R	966	push seg Exception_handler
01D9 68290290		967	push offset Exception_handler
01D0 9AE701----	R	968	call far ptr TQSETERH ; set up the exception handle
		969	r
01E2 B80000		970	mov AX, E_ok
01E5 1F		971	pop DS ; clean up
01E6 CA0400		972	ret 4 ; and go
		973	
		974	TQINITIALIZE endp
		975	
		976	public TQSETERH
		977	
		978	
		979 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 31

```

LDC DBJ          LINE  SOURCE
                  980  ;-----
                  981  ;
01E9             982  T0SETERH      proc      far
                  983  ;
                  984  ; FUNCTION : Sets the address of the current exception handler.
                  985  ;
                  986  ; INPUTS : Address of the current exception handler.
                  987  ;
                  988  ; CALLS : None.
                  989  ;
                  990  ; OUTPUTS : Status in AX.
                  991  ;           Address of the new exception handler in Exception_addr.
                  992  ;
                  993  ; DESTROYS : Flags.
                  994  ;
                  995  ;
                  996  assume      DS : IO_data
                  997
01E9 C8000000     998              enter    0, 0
01ED 1E          999              push     DS
01EE 88-----   R 1000             mov      AX, IO_data
01F1 8ED8        1001             mov      DS, AX              ; set up for access to
                  1002                                     ; local data
                  1003
01F3 8B4606      1004             mov      AX, word ptr [BP + 6] ; offset
01F6 A38700      1005             mov      word ptr Exception_addr, AX
01F9 8B4608      1006             mov      AX, word ptr [BP + 8] ; segment
01FC A38900      1007             mov      word ptr Exception_addr + 2, AX
                  1008
01FF 8B0000      1009             mov      AX, E_ok
0202 1F          1010             pop      DS              ; clean up
0203 C9          1011             leave
0204 CA0400      1012             ret      4              ; and go
                  1013
                  1014 T0SETERH      endp
                  1015
                  1016
                  1017 public T0SETERH
                  1018
1019 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 32

LOC	OBJ	LINE	SOURCE
		1020	;-----
		1021	
0207		1022	TQETERM proc far
		1023	;
		1024	; FUNCTION : Returns the address of the current exception handler, as
		1025	; set by TQETERM.
		1026	;
		1027	; INPUTS : Address to which the address of the exception handler must be
		1028	; written in [BP + 6].
		1029	;
		1030	; CALLS : None.
		1031	;
		1032	; OUTPUTS : Status in AX.
		1033	;
		1034	; DESTROYS : AX, DI, ES, Flags.
		1035	;
		1036	;
		1037	assume DS : IO_data
		1038	
0207 C8000000		1039	enter 0, 0
020B 1E		1040	push DS
020C B8----	R	1041	mov AX, IO_data
020F 8ED8		1042	mov DS, AX ; set up for access to
		1043	; local data
		1044	
0211 C47E06		1045	les DI, dword ptr [BP + 6] ; address pointer
0214 A18700		1046	mov AX, word ptr Exception_addr
0217 268905		1047	mov word ptr ES:[DI], AX ; the first part
021A A18900		1048	mov AX, word ptr Exception_addr + 2
021D 26894502		1049	mov word ptr ES:[DI + 2], AX ; the second part
		1050	
0221 880000		1051	mov AX, E_ok
		1052	
0224 1F		1053	pop DS ; clean up
0225 C9		1054	leave
0226 CA0400		1055	ret 4 ; and go
		1056	
		1057	TQETERM endp
		1058	
		1059	assume DS : nothing
		1060	
		1061 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 33

LOC	OBJ	LINE	SOURCE
		1062	-----
		1063	
0229		1064	Exception_handler proc far
		1065	;
		1066	; FUNCTION : This procedure is called in the event of a Pascal-86 exception.
		1067	; It outputs the exception parameters to conl and then resets
		1068	; the system..
		1069	;
		1070	; INPUTS : Exception code in [BP + 12].
		1071	; Parameter number in [BP + 10].
		1072	; Numeric exception code in [BP + 6].
		1073	;
		1074	; CALLS : Conout, Out_AX.
		1075	;
		1076	; OUTPUTS : None.
		1077	;
		1078	; DESTROYS : N/A.
		1079	;
		1080	;
		1081	assume DS : Tables
		1082	
0229	C8000000	1083	enter 0, 0
022D	1E	1084	push DS
022E	88----	1085	mov AX, Tables
0231	8ED8	1086	mov DS, AX ; set up for access to
		1087	; local data
0233	FC	1088	cld ; Auto increment.
		1089	
		1090 +1	
0234	B92500	1091 +2	mov CX, length Err_msg_1 ; length of the string
0237	BE6600	1092 +2	mov SI, offset Err_msg_1 ; offset of the string
023A	AC	1093 +2	ST_LB_02 : lods Err_msg_1
023B	E8BE00	1094 +1	call Conout ; output the character
023E	E2FA	1095 +2	loop ST_LB_02 ; until finished
		1096 +1	
0240	8B460C	1097	mov AX, word ptr [BP + 12] ; Exception code
0243	E88200	1098	call Out_AX
		1099 +1	
0246	B92600	1100 +2	mov CX, length Err_msg_2 ; length of the string
0249	BEBB00	1101 +2	mov SI, offset Err_msg_2 ; offset of the string
024C	AC	1102 +2	ST_LB_03 : lods Err_msg_2
024D	E8AC00	1103 +1	call Conout ; output the character
0250	E2FA	1104 +2	loop ST_LB_03 ; until finished

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 34

LOC	OBJ	LINE	SOURCE
		1105 +1	
0252	8B460A	1106	mov AX, word ptr [BP + 10] ; Parameter number
0255	E87000	1107	call Out_AX
		1108 +1	
0258	892600	1109 +2	mov CX, length Err_msg_3 ; length of the string
025B	BE8100	1110 +2	mov SI, offset Err_msg_3 ; offset of the string
025E	AC	1111 +2	ST_LB_04 : lods Err_msg_3
025F	E89A00	1112 +1	/call Conout ; output the character
0262	E2FA	1113 +2	loop ST_LB_04 ; until finished
		1114 +1	
0264	8B4606	1115	mov AX, word ptr [BP + 6] ; Numeric exception code
0267	E85E00	1116	call Out_AX
		1117 +1	
026A	892600	1118 +2	mov CX, length Err_msg_4 ; length of the string
026D	BE8700	1119 +2	mov SI, offset Err_msg_4 ; offset of the string
0270	AC	1120 +2	ST_LB_05 : lods Err_msg_4
0271	E88800	1121 +1	call Conout ; output the character
0274	E2FA	1122 +2	loop ST_LB_05 ; until finished
		1123 +1	
0276	8B4604	1124	mov AX, word ptr [BP + 4] ; Segment of the ret
0279	E84C00	1125	call Out_AX
027C	803A	1126	mov AL, ':'
027E	E87800	1127	call Conout
0281	8B4602	1128	mov AX, word ptr [BP + 2] ; Offset
0284	E84100	1129	call Out_AX
		1130 +1	
0287	890300	1131 +2	mov CX, length Err_msg_5 ; length of the string
028A	BEFD00	1132 +2	mov SI, offset Err_msg_5 ; offset of the string
028D	AC	1133 +2	ST_LB_06 : lods Err_msg_5
028E	E86B00	1134 +1	call Conout ; output the character
0291	E2FA	1135 +2	loop ST_LB_06 ; until finished
		1136 +1	
		1137	
		1138	
0293	FAEA0000FFFF	1139	reset
		1140	
		1141	Exception_handler endp
		1142	
		1143	assume DS : nothing
		1144	
		1145 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 35

```

LOC OBJ          LINE    SOURCE
1146 ;*****
1147 ;*****
1148 ;
1149 ; LOCAL PROCEDURES
1150 ;
1151 ;*****
1152 ;*****
1153 ;
1154 ;
1155 ;-----
1156
0299 1157 Get_line      proc      near
1158 ;
1159 ; FUNCTION : Reads a new line into the local buffer for the LRS's read
1160 ;           function.
1161 ;
1162 ; INPUTS : None.
1163 ;
1164 ; CALLS : Conin, Conout.
1165 ;
1166 ; OUTPUTS : The new line in the local buffer.
1167 ;           The local buffer index updated.
1168 ;
1169 ; DESTROYS : AX, BX, DX, SI, Flags.
1170 ;
1171 ;
1172
1173 assume         DS : IO_data
1174
1175
0299 BE0000 1176          mov     SI, 0           ; zero the count
029C 81FE8400 1177 Get_more :    cmp     SI, length Local_buffer ; check for buffer overflow
02A0 7201      1178          jb      Still_room
02A2 4E        1179          dec     SI           ; if so then delete a character
02A3 E83F00   1180 Still_room :  call    Conin
02A6 E85300   1181          call    Conout        ; echo the character
02A9 3C08     1182          cmp     AL, 85
02AB 7404     1183          je      Back_sp      ; if we encounter a back-space
02AD 3C7F     1184          cmp     AL, DEL      ; or a del goto Back_sp
02AF 7503     1185          jne     Store        ; the character can be stored
02B1 4E       1186 Back_sp :    dec     SI           ; delete the previous character
02B2 EBE8     1187          jmp     Get_more      ; and go back for more
1188

```

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 36

```

LSC OBJ          LINE  SOURCE
02B4 884402      1189  Store :      mov     Local_buffer[SI], AL    ; store the character
02B7 46          1190          inc     SI                      ;
02B8 3C00        1191          cmp     AL, CR                ; CR character?
02BA 75E0        1192          jne     Get_more             ; if not then go look for
                                1193                      ; some more characters
02BC 800A        1194          mov     AL, LF                ;
02BE E83800      1195          call    Conout             ; add an LF
02C1 C70600000000 1196          mov     Local_index, 0        ; zero the local index
02C7 C3          1197          ret                      ; and go
                                1198
                                1199
                                1200  Get_line      endp
                                1201
                                1202
                                1203
                                1204
                                1205
                                1206 +1 $EJECT

```


8086/87/58/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 37

```

LOC OBJ          LINE    SOURCE
                  1207    ;-----
                  1208
02C3             1209    Out_AX      proc    near
                  1210    ;
                  1211    ; FUNCTION : Outputs the hex word in AX as a string of ASCII characters.
                  1212    ;
                  1213    ; INPUTS : Word in AX.
                  1214    ;
                  1215    ; CALLS : Asc_out.
                  1216    ;
                  1217    ; OUTPUTS : None.
                  1218    ;
                  1219    ; DESTROYS : AX, BX, CX, Flags.
                  1220    ;
                  1221    ;
                  1222
                  1223
                  1224
02C8 8BC8        1225    mov     CX, AX                ; Save the word
02CA 8AC5        1226    mov     AL, CH                ; Get the high byte
02CC C0E804      1227    shr     AL, 4                ; Get the high nibble
02CF E82000      1228    call    Asc_out                ; Output it
02D2 8AC5        1229    mov     AL, CH
02D4 E81B00      1230    call    Asc_out                ; Output the low nibble
02D7 8AC1        1231    mov     AL, CL                ; Get the low byte
02D9 C0E804      1232    shr     AL, 4                ; Get the high nibble
02DC E81300      1233    call    Asc_out
02DF 8AC1        1234    mov     AL, CL
02E1 E80E00      1235    call    Asc_out                ; Output the low nibble
                  1236
02E4 C3          1237    ret                        ; and go
                  1238
                  1239
                  1240    Out_AX      endp
                  1241
                  1242
                  1243
                  1244
                  1245
                  1246
1247 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 38

LOC OBJ

LINE SOURCE

```

1248 ;-----
1249
02E5 1250 Conin      proc      near
1251
1252 ; FUNCTION : This procedure returns a character recieved on USART1.
1253 ;
1254 ; INPUTS : None
1255 ;
1256 ; CALLS : None.
1257 ;
1258 ; OUTPUTS : Character in AL
1259 ;
1260 ; DESTROYS : AX, DX, flags.
1261 ;
1262
02E5 8A8204 1263      mov     DX, USART1_control    ;Get USART status
02E8 EC    1264      Look :    in      AL, DX
02E9 A802  1265      test   AL, Rec_mask    ;Ready?
02EB 74FB  1266      jz     Look           ; if not ready then try again
02ED 8A8004 1267      mov     DX, USART1_data
02F0 EC    1268      in      AL, DX      ; get the character
02F1 C3    1269      ret
1270
1271      Conin      endp
1272
1273
1274
1275
1276
1277
1278 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 39

```

LOC OBJ      LINE  SOURCE
1279  ;-----
1280
02F2 1281  Asc_out      proc      near
1282
1283  ; FUNCTION : This procedure outputs the ASCII value corresponding to the
1284  ;             low nibble in AL on USART1. Note that this procedure falls
1285  ;             through to Conout.
1286  ;
1287  ; INPUTS : Value in AL
1288  ;
1289  ; CALLS : None.
1290  ;
1291  ; OUTPUTS : None
1292  ;
1293  ; DESTROYS : AX, BX, DX, flags
1294  ;
1295
1296
02F2 240F 1297          and      AL, 0Fh      ; Mask off the low nibble
02F4 0430 1298          add      AL, 30h      ; Adjust to ASCII
02F6 3C39 1299          cmp      AL, 39h      ; Do we need to adjust?
02F8 7602 1300          jbe      Conout
02FA 0407 1301          add      AL, 7       ; To get A, B etc
1302
1303 +1. $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

PASCAL86_LOGICAL_RECORD_SYSTEM

04/29/86 PAGE 40

```

LOC DBJ      LINE    SOURCE
              1304
02FC          1305      Conout      proc      near
              1306
              1307      ; FUNCTION : This procedure outputs a character to USART1.
              1308      ;
              1309      ; INPUTS : Character in AL
              1310      ;
              1311      ; CALLS : None.
              1312      ;
              1313      ; OUTPUTS : None
              1314      ;
              1315      ; DESTROYS : BX, DX, flags
              1316      ;
              1317
02FC 8B08     1318          mov      BX, AX          ; Save the character
02FE BA8204   1319          mov      DX, USART1_control
0301 EC       1320      Look2 :      in      AL, DX          ; Input the status
0302 AB01     1321          test     AL, Tr_mask      ; Ready?
0304 74FB     1322          jz      Look2          ; if not, else...
0306 BA8004   1323          mov      DX, USART1_data
0309 8BC3     1324          mov      AX, BX          ; get the character back
030B EE       1325          out      DX, AL
030C C3       1326          ret
              1327
              1328      Conout      endp
              1329
              1330
              1331      Asc_out      endp
              1332
----         1333      ID_code ends
              1334
              1335          end

```

ASSEMBLY COMPLETE, NO ERRORS FOUND

8086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 1

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE TAPE_IO_CONTROL_MODULE

OBJECT MODULE PLACED IN :F3:ADMTIO.OBJ

ASSEMBLER INVOKED BY: ASM86.86 :F3:ADMTIO.ASM

LOC	OBJ	LINE	SOURCE
-----	-----	------	--------

		1 +1	\$EJECT
--	--	------	---------

8086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 2

```

LOC 08:      LINE    SOURCE
              2 +1 $DEBUG
              3 +1 $GENONLY
              4 +1 $TYPE
              5 +1 $PAGELENGTH(48)
              6 +1 $MCD186
              7      ;
              8      ;*****
              9      ;*****
             10      ;
             11      ;          MODULE FUNCTION :
             12      ;
             13      ; This module provides primitives used to control nine channel tape
             14      ; transports.
             15      ;
             16      ;*****
             17      ;
             18      ;      Written : A.O.McGuffog          14 October 1985.
             19      ;
             20      ;*****
             21      ;
             22      ;      Language : Intel ASM 86.
             23      ;
             24      ;*****
             25      ;
             26      ;      Other Software Required : Designed to run with Intel Pascal-86 V3
             27      ;
             28      ;*****
             29      ;
             30      ;      Hardware required : 80188 CPU card , tape controller module.
             31      ;
             32      ;*****
             33      ;
             34      ;      Restrictions : None.
             35      ;
             36      ;*****
             37      ;
             38      ;          Module description :
             39      ;
             40      ;The Tape controller software provides several services. These are :
             41      ;
             42      ;      1. A unit may be selected, and its timing parameters set. All subsequent
             43      ;      operations refer to the selected unit.
             44      ;

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 3

LOC	OBJ	LINE	SOURCE
		45	;
		46	;
		47	;
		48	3. A block may be written to tape with either a normal or long inter-record
		49	gap.
		50	;
		51	;
		52	4. An end-of-file mark may be written.
		53	;
		54	;
		55	5. A block may be read from tape.
		56	;
		57	;
		58	6. The tape may be spaced either forward or back by a specified number
		59	of blocks.
		60	;
		61	;
		62	7. The status of the selected unit may be read.
		63	;
		64	8. Obtain diagnostics information.
		65	;
		66	;
		67	9. The tape controller subsystem may be initialized.
		68	;
		69	;
		70	;
		71	;
		72	;
		73	;
		74	;
		75	\$INCLUDE(:F3:ADNDEF.ASM)
		76	;
		77	;
		78	;
		79	;
		80	;
		81	;
		82	;
		83	;
		84	;
		85	;
		86	;
		87	;
		88	;
		89	;
		90	;
		91	;
		92	;
		93	;
		94	;
		95	;
		96	;
		97	;
		98	;
		99	;
		100	;
		101	;
		102	;
		103	;
		104	;
		105	;
		106	;
		107	;
		108	;
		109	;
		110	;
		111	;
		112	;
		113	;
		114	;
		115	;
		116	;
		117	;
		118	;
		119	;
		120	;
		121	;
		122	;
		123	;
		124	;
		125	;
		126	;
		127	;
		128	;
		129	;
		130	;
		131	;
		132	;
		133	;
		134	;
		135	;
		136	;
		137	;
		138	;
		139	;
		140	;
		141	;
		142	;
		143	;
		144	;
		145	;
		146	;
		147	;
		148	;
		149	;
		150	;
		151	;
		152	;
		153	;
		154	;
		155	;
		156	;
		157	;
		158	;
		159	;
		160	;
		161	;
		162	;
		163	;
		164	;
		165	;
		166	;
		167	;
		168	;
		169	;
		170	;
		171	;
		172	;
		173	;
		174	;
		175	;
		176	;
		177	;
		178	;
		179	;
		180	;
		181	;
		182	;
		183	;
		184	;
		185	;
		186	;
		187	;
		188	;
		189	;
		190	;
		191	;
		192	;
		193	;
		194	;
		195	;
		196	;
		197	;
		198	;
		199	;
		200	;
		201	;
		202	;
		203	;
		204	;
		205	;
		206	;
		207	;
		208	;
		209	;
		210	;
		211	;
		212	;
		213	;
		214	;
		215	;
		216	;
		217	;
		218	;
		219	;
		220	;
		221	;
		222	;
		223	;
		224	;
		225	;
		226	;
		227	;
		228	;
		229	;
		230	;
		231	;
		232	;
		233	;
		234	;
		235	;
		236	;
		237	;
		238	;
		239	;
		240	;
		241	;
		242	;
		243	;
		244	;
		245	;
		246	;
		247	;
		248	;
		249	;
		250	;
		251	;
		252	;
		253	;
		254	;
		255	;
		256	;
		257	;
		258	;
		259	;
		260	;
		261	;
		262	;
		263	;
		264	;
		265	;
		266	;
		267	;
		268	;
		269	;
		270	;
		271	;
		272	;
		273	;
		274	;
		275	;
		276	;
		277	;
		278	;
		279	;
		280	;
		281	;
		282	;
		283	;
		284	;
		285	;
		286	;
		287	;
		288	;
		289	;
		290	;
		291	;
		292	;
		293	;
		294	;
		295	;
		296	;
		297	;
		298	;
		299	;
		300	;
		301	;
		302	;
		303	;
		304	;
		305	;
		306	;
		307	;
		308	;
		309	;
		310	;
		311	;
		312	;
		313	;
		314	;
		315	;
		316	;
		317	;
		318	;
		319	;
		320	;
		321	;
		322	;
		323	;
		324	;
		325	;
		326	;
		327	;
		328	;
		329	;
		330	;
		331	;
		332	;
		333	;
		334	;
		335	;
		336	;
		337	;
		338	;
		339	;
		340	;
		341	;
		342	;
		343	;
		344	;
		345	;
		346	;
		347	;
		348	;
		349	;
		350	;
		351	;
		352	;
		353	;
		354	;
		355	;
		356	;
		357	;
		358	;
		359	;
		360	;
		361	;
		362	;
		363	;
		364	;
		365	;
		366	;
		367	;
		368	;
		369	;
		370	;
		371	;
		372	;
		373	;
		374	;
		375	;
		376	;
		377	;
		378	;
		379	;
		380	;
		381	;
		382	;
		383	;
		384	;
		385	;
		386	;
		387	;
		388	;
		389	;
		390	;
		391	;
		392	;
		393	;
		394	;
		395	;
		396	;
		397	;
		398	;
		399	;
		400	;
		401	;
		402	;
		403	;
		404	;
		405	;
		406	;
		407	;
		408	;
		409	;
		410	;
		411	;
		412	;
		413	;
		414	;
		415	;
		416	;
		417	;
		418	;
		419	;
		420	;
		421	;
		422	

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 4

LOC	OBJ	LINE	SOURCE
FF54		=1 88	MCNT0B_reg equ Per_base + 054h
FF56		=1 89	TMR0CNTR_reg equ Per_base + 056h
FF58		=1 90	CNT1_reg equ Per_base + 058h
FF5A		=1 91	MCNT1A_reg equ Per_base + 05Ah
FF5C		=1 92	MCNT1B_reg equ Per_base + 05Ch
FF5E		=1 93	TMR1CNTR_reg equ Per_base + 05Eh
FF60		=1 94	CNT2_reg equ Per_base + 060h
FF62		=1 95	MCNT2A_reg equ Per_base + 062h
FF66		=1 96	TMR2CNTR_reg equ Per_base + 066h
		=1 97	
		=1 98	;Now the memory chip select registers
		=1 99	
FFA0		=1 100	UMCS_reg equ Per_base + 0A0h ;Upper select
FFA2		=1 101	LMCS_reg equ Per_base + 0A2h ;Lower select
FFA4		=1 102	PACS_reg equ Per_base + 0A4h ;Peripheral select
FFA6		=1 103	MMCS_reg equ Per_base + 0A6h ;Memory select
FFA8		=1 104	MPCS_reg equ Per_base + 0A8h
		=1 105	
FFCA		=1 106	DMA_CO_reg equ Per_base + 0CAh ;DMA control registers
FFDA		=1 107	DMA_C1_reg equ Per_base + 0DAh
FFC8		=1 108	DMA_T0_reg equ Per_base + 0C8h ;DMA transfer count registers
FFD8		=1 109	DMA_T1_reg equ Per_base + 0D8h
FFC6		=1 110	DMA_DPH0_reg equ Per_base + 0C6h ;DMA destination high and low
FFD6		=1 111	DMA_DPH1_reg equ Per_base + 0D6h
FFC4		=1 112	DMA_DPLO_reg equ Per_base + 0C4h
FFD4		=1 113	DMA_DPL1_reg equ Per_base + 0D4h
FFC2		=1 114	DMA_SPH0_reg equ Per_base + 0C2h ;DMA source high and low
FFD2		=1 115	DMA_SPH1_reg equ Per_base + 0D2h
FFC0		=1 116	DMA_SPLO_reg equ Per_base + 0C0h
FFD0		=1 117	DMA_SPL1_reg equ Per_base + 0D0h
		=1 118	
		=1 119	; Now the interrupt control registers :
		=1 120	
FF22		=1 121	INT_EOI_reg equ Per_base + 022h ;End of interrupt register
FF24		=1 122	INT_POL_reg equ Per_base + 024h ;Poll register
FF26		=1 123	INT_PST_reg equ Per_base + 026h ;
FF28		=1 124	INT_MSK_reg equ Per_base + 028h ;Mask register
FF2A		=1 125	INT_PRN_reg equ Per_base + 02Ah ;Priority register
FF2C		=1 126	INT_ISR_reg equ Per_base + 02Ch ;In service register
FF2E		=1 127	INT_REQ_reg equ Per_base + 02Eh ;Request register
FF30		=1 128	INT_STS_reg equ Per_base + 030h ;Status
FF32		=1 129	INT_TMC_reg equ Per_base + 032h ;Timer interrupt

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 5

LOC	OBJ	LINE	SOURCE
FF34		=1 130	INT_DMA0_reg equ Per_base + 034h ;DMA interrupts
FF36		=1 131	INT_DMA1_reg equ Per_base + 036h
FF38		=1 132	INT_INT0_reg equ Per_base + 038h ;External interrupts
FF3A		=1 133	INT_INT1_reg equ Per_base + 03Ah
FF3C		=1 134	INT_INT2_reg equ Per_base + 03Ch
FF3E		=1 135	INT_INT3_reg equ Per_base + 03Eh
		=1 136	
		=1 137	
FE3F		=1 138	UMCS_val equ 0FE3Fh ;Values for the memory registers
3FFB		=1 139	LMCS_val equ 03FFBh ; 4 by 2764s, 512k ram,
007F		=1 140	PACS_val equ 0007Fh ; 3 wait states + ext, peripheral
F9FF		=1 141	NMCS_val equ 0F9FFh ; select block in IO space,
843F		=1 142	MPCS_val equ 0843Fh ; 3 IO wait states + ext.
		=1 143	
8000		=1 144	NS_eoi equ 1000000000000000b ;Non specific end of interrupt
		=1 145	
0005		=1 146	Maxcount0a equ 5 ;Count values for default 9600
0005		=1 147	Maxcount0b equ 5 ; baud serial lines
0005		=1 148	Maxcount1a equ 5
0005		=1 149	Maxcount1b equ 5
C003		=1 150	TMR_baud_control equ 1100000000000011b
		=1 151	;
		=1 152	; Now the USART equates :
		=1 153	
0400		=1 154	PCS_base equ 0400h ;The two USART's on the CPU card
0400		=1 155	USART0_data equ PCS_base
0402		=1 156	USART0_control equ PCS_base + 2
0480		=1 157	USART1_data equ PCS_base + 128
0482		=1 158	USART1_control equ PCS_base + 130
00CA		=1 159	USART_mode_7 equ 11001010b ; 2 stop bits, no parity, 7 bits
00CE		=1 160	USART_mode_8 equ 11001110b ; 2 stop bits, no parity, 8 bits
0037		=1 161	USART_cmd equ 00110111b ; Enable all
0040		=1 162	USART_reset equ 01000000b
000A		=1 163	Dummy_cmd equ 00001010b
0002		=1 164	Rec_mask equ 02h ; Look for RE ready
0001		=1 165	Tr_mask equ 01h ; Look for TX ready
0000		=1 166	CR equ 0Dh
000A		=1 167	LF equ 0Ah
		=1 168	
		=1 169	;Bit defs for the bits in the interrupt mask register
		=1 170	
#		=1 171	Int_bits record R1_int:1,T1_int:1,R0_int:1,T0_int:1,
		=1 172	& DMA_1_int:1,DMA_0_int:1,Int_bits_dummy:1,TMR_int:1

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 6

```

LOC 0807          LINE    SOURCE
                    =1 173
                    =1 174      ;Bit defs for the 8251 USART status register
                    =1 175
#                  =1 176      USART_bits      record  OSR:1,SYN_det:1,USART_error:3,TX_empty:1,RX_rdy:1,
                    =1 177      &                TX_rdy:1
                    178      ;
                    179      ;
                    180      name    Tape_ID_control_module
                    181      ;
                    182      ;
                    183      ;Set up a stack segment:
                    184      ;
-----          185      stack      segment          STACK          'STACK'
                    186
0000 (50          187              dw      50          dup      (?)
      ????)
      )

                    188
-----          189      stack      ends
                    190      ;
                    191      ;
                    192      ;Now some equates
                    193      ;
                    194      ;
0800          195      Tape_base      equ      0800h          ;Base address of the controller
                    196
0806          197      Tmr_0_cntrl      equ      Tape_base + 0110b          ;First 8254
0800          198      Start_c      equ      Tape_base + 0000b          ;Start gap timer
0802          199      Event_c      equ      Tape_base + 0010b          ;Event counter
0804          200      Stop_c      equ      Tape_base + 0100b          ;Stop gap timer
                    201
080E          202      Tmr_1_cntrl      equ      Tape_base + 01110b          ;Second 8254
0808          203      Master_c      equ      Tape_base + 01000b          ;Character clock divider
080A          204      L5_det      equ      Tape_base + 01010b          ;Long gap timer
080C          205      S6_det      equ      Tape_base + 01100b          ;Short gap timer
                    206
0810          207      Write_data      equ      Tape_base + 010000b          ;Data out port
0818          208      Write_status      equ      Tape_base + 011000b          ;Status out port
0820          209      Write_aux      equ      Tape_base + 0100000b          ;Parity, strobe enable
0828          210      Write_cmd      equ      Tape_base + 0101000b          ;Command bit port
                    211
0830          212      Read_aux      equ      Tape_base + 0110000b          ;Parity, state in

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 7

LOC	OBJ	LINE	SOURCE
0834		213	Read_status equ Tape_base + 0110100b ;Status in port
0838		214	Read_data equ Tape_base + 0111000b ;Data in port
08D2		215	Read_back equ Tape_base + 011010010b ;8254 Timer read back
		216	
		217	;Now some commands:
		218	
		219	
0001		220	Go_cmd equ 0001b ;Go operation
0000		221	Clear_go_cmd equ 0000b
0003		222	Set_write_status equ 0011b ;Write enable
0002		223	Clear_write_status equ 0010b
0005		224	Rewind_cmd equ 0101b ;Rewind selected unit
0007		225	Off_line_cmd equ 0111b ;Set unit off-line
0009		226	RTH_high_cmd equ 1001b ;Read threshold high
0008		227	RTH_low_cmd equ 1000b ;Read threshold low
000A		228	Board_reset equ 1010b ;Reset controller
0008		229	Clear_board_reset equ 1011b
		230	
		231	
0010		232	Write_status_val equ 00010000b ;Status value for write
0034		233	Read_status_val equ 00110100b ;Status value for read
0014		234	Erase_status_val equ 00010100b ;Status value for erase
0030		235	Space_fwd_val equ 00110000b ;Status value for fwd space
0020		236	Space_rev_val equ 00100000b ;Status value for rev space
0004		237	Count_chars equ 0100b ;Status value for char count
0008		238	Count_blocks equ 1000b ;Status value for block count
		239	
0032		240	Start_c_init equ 000110010b ;8254 initialization values
00B2		241	Stop_c_init equ 010110010b
0072		242	Event_c_init equ 001110010b
0034		243	Master_c_init equ 000110100b
0072		244	LG_det_init equ 001110010b
00B2		245	SG_det_init equ 010110010b
0070		246	Event_c_space_mode equ 001110000b
		247	
00FF		248	True equ 0FFh
0000		249	False equ 00h
		250	
		251	
0032		252	Run_delay equ 50 ;Delay until run bit valid
1F40		253	Tape_settle_delay equ 8000 ;Delay till tape motion
		254	; ceases
		255	

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 8

LOC	OBJ	LINE	SOURCE
02C8		256	Tape_mark equ 01011001000b ;Tape mark data and parity
		257	
000E		258	State_mask equ 01110b ;State value extraction
0006		259	State_3 equ 00110b
		260	
		261	
01FF		262	Invert_mask equ 011111111b ;Bits to be inverted in data
		263	
0003		264	Unit_sel_mask equ 011b ;Unit select bits
		265	
013C		266	CRC_feedback equ 00100111100b ;Tape CRC values
01D7		267	CRC_final equ 0111010111b
		268	
#		269	error_flags record PEDV:1,EDV:1,BOT:1,Time_err:1,xx:1,HWL:1,RWD:1,S_err:1,ISX:1,
		270	& SWL:1,EOF:1,EDT:1,DMA_err:1,Length_err:1,CRC_err:1,
		271	& P_err:1
		272	
		273	New_err_bits equ mask P_err or mask CRC_err or mask
100F		274	& Length_err or mask DMA_err or mask Time_err
		275	
		276	New_status_bits equ mask EDT or mask HWL or mask BOT or mask EDV
6710		277	& or mask S_err or mask RWD
		278	
0100		279	Parity_mask equ 0100h
		280	
773F		281	Change_bits equ New_err_bits or New_status_bits or mask EOF
		282	
#		283	In_flags record EOF_in:1,RUN:1,RWD_in:1,FPT_in:1,EDT_in:1,LDP_in:1,
		284	& DN_LINE_in:1,RDY_in:1
		285	
#		286	Cntr_status_bits record Output:1, Null_count:1, RW1:1, RW0:1, M2:1,
		287	& M1:1, M0:1, BCD:1
		288	
2000		289	max_block_length equ 2000h
		290	
A246		291	ID_source equ 1010001001000110b ;DMA reg setup values
1686		292	ID_dest equ 0001011010000110b
1684		293	Stop_DMA equ 0001011010000100b
		294	
0002		295	WDS_en equ 0010b ;Strobe enable values
0004		296	WARS_en equ 0100b
0000		297	Nothing_en equ 0000b
		298	

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 9

LOC	OBJ	LINE	SOURCE
0009		299	Extra_write_bytes equ 9 ;Number of extra bytes
		300	; for DMA output
		301	
0007		302	EOF_data_gap equ 7 ;Gap between data bytes
		303	; in tape mark
		304	
01FF		305	CH_9 equ 0111111111b
		306	
0010		307	Over_run_mask equ 010000b ;Get the data over-run bit
		308	
00C4		309	Read_back_cmd equ 011000100b ;8254 read back cmd
00E4		310	Read_back_L6 equ 011100100b
		311	
		312	
		313	;Now for some data!
		314	
----		315	T_data segment 'DATA'
		316	
0000 ??		317	Out_status db ?
0001 ????		318	Stop_norm dw ?
0003 ????		319	Start_norm dw ?
0005 ????		320	Start_long dw ?
0007 ????		321	Start_eof dw ?
0009 ????		322	Start_read dw ?
000B ????		323	Error dw ?
000D ????		324	Data_1 dw ?
000F ????		325	CRC dw ?
0011 ????		326	LRC dw ?
		327	
		328	
0013 (8202		329	Local_buf db max_block_length + 10 dup (?)
??			
)			
2010 (10		330	EOF_buf db 10 dup (?)
??			
)			
		331	
----		332	T_data ends
		333	
----		334	T_code segment 'CODE'
		335	
		336	assume CS : T_code
		337	assume DS : T_data

8086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 10

LOC OBJ

LINE SOURCE

338

339 +1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 11

LOC OBJ

LINE

SOURCE

```

340 ;-----
341 ;Procedure Select
342 ;
343 ; FUNCTION : Selects a unit and sets it's timing parameters.
344 ;
345 ; INPUTS : Note all values in 200 nS increments.
346 ;
347 ;     Stop_val      : Count till command inactive from last data
348 ;     MC_val        : Character clock divisor
349 ;     LG_val        : Count to detect a long gap
350 ;     SG_val        : Count to detect a short gap
351 ;     Start_norm_val : Count from motion start to first data
352 ;     Start_long_val : Count from motion start to first data with long gap
353 ;     Start_EOF_val  : Count from motion start to first data for EOF mark
354 ;     Start_read_val : Count from motion start till first valid data possible
355 ;                     for read operation
356 ;     Unit           : Unit number
357 ;
358 ; CALLS : None.
359 ;
360 ; OUTPUTS : None.
361 ;
362 ; DESTROYS : AX, DX, Flags.
363 ;
364 ;
365
-----
366 select_struct  struc
367
0000 368     sel_old_BP    dw    ?
0002 369     sel_return   dd    ?
0006 370     Stop_val     dw    ?
0008 371     MC_val      dw    ?
000A 372     LG_val      dw    ?
000C 373     SG_val      dw    ?
000E 374     Start_norm_val dw  ?
0010 375     Start_long_val dw  ?
0012 376     Start_EOF_val dw  ?
0014 377     Start_read_val dw  ?
0016 378     Unit         dw    ?
379
-----
380 select_struct  ends
381
382

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 12

LOC	OBJ	LINE	SOURCE
		383	public Select
		384	
0000		385	Select proc far
		386	
		387	
		388	
0000	C8000000	389	enter 0, 0
0004	1E	390	push DS
0005	B8----	391	mov AX, T_data
0008	8ED8	392	mov DS, AX ;Now we can get on with it
000A	81260B00C088	393	and Error, not Change_bits
0010	8B4606	394	mov AX, [BP].Stop_val ;Get the stop value
0013	A30100	395	mov Stop_norm, AX ; and store it
0016	8A0808	396	mov DX, Master_c ;Get ready to set up the clock
0019	8B4608	397	mov AX, [BP].MC_val
001C	EE	398	out DX, AL ;First the lsb
001D	8AC4	399	mov AL, AH
001F	EE	400	out DX, AL ; then the msb
0020	8A0A08	401	mov DX, L6_det ;Now do the same for the L6 counter
0023	8B460A	402	mov AX, [BP].L6_val
0026	EE	403	out DX, AL
0027	8AC4	404	mov AL, AH
0029	EE	405	out DX, AL
002A	8A0C08	406	mov DX, S6_det ;Now the S6 counter
002D	8B460C	407	mov AX, [BP].S6_val
0030	EE	408	out DX, AL
0031	8AC4	409	mov AL, AH
0033	EE	410	out DX, AL
0034	8A1808	411	mov DX, Write_status ;Now get ready to set up the
		412	; controller
0037	8B4616	413	mov AX, [BP].Unit ;Set the unit number
003A	250300	414	and AX, Unit_sel_mask ; and only the unit number
003D	A20000	415	mov Out_status, AL ;Store that info
0040	EE	416	out DX, AL ;Tell the controller about it
0041	8B460E	417	mov AX, [BP].Start_norm_val
0044	A30300	418	mov Start_norm, AX
0047	8B4610	419	mov AX, [BP].Start_long_val
004A	A30500	420	mov Start_long, AX
004D	8B4612	421	mov AX, [BP].Start_EOF_val
0050	A30700	422	mov Start_EOF, AX
0053	8B4614	423	mov AX, [BP].Start_read_val
0056	A30900	424	mov Start_read, AX
		425	

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 13

LOC	OBJ	LINE	SOURCE
0059	BA2808	426	mov 0X, Write_cmd ;Ok, now get the board back
		427	; to normal
005C	B00A	428	mov AL, Board_reset
005E	EE	429	out DX, AL
005F	B002	430	mov AL, Clear_write_status ;Reset write status
0061	EE	431	out DX, AL
		432	
0062	1F	433	pop DS ;That's all for now
0063	C9	434	leave
0064	CA1200	435	ret size select_struct - 6
		436	
		437	Select endp
		438	
		439	
		440	+1 \$EJECT

8066/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 14

LOC	OBJ	LINE	SOURCE
		441	;
		442	;-----
		443	;Procedure Erase_tape
		444	;
		445	; FUNCTION : Erases the tape from the current position to the EOT marker.
		446	;
		447	; INPUTS : None.
		448	;
		449	; CALLS : Set_start, Cmd_reset, Clean_timers.
		450	;
		451	; OUTPUTS : None.
		452	;
		453	; DESTROYS : AX, BX, DX, CX, Flags.
		454	;
		455	;
		456	public Erase_tape
0067		457	Erase_tape proc far
		458	
		459	
		460	
0067	C8000000	461	enter 0, 0
006B	1E	462	push DS
006C	88----	463	mov AX, T_data ;Get ready for data access
006F	8ED8	464	mov DS, AX
0071	A10300	465	mov AX, Start_norm ;Get ready to set up the start
0074	E8A304	466	call Set_start ;Go do it
		467	
0077	8A1808	468	mov DX, Write_status ;Now set up the controller
007A	A00000	469	mov AL, Out_status ;Retrieve the unit number etc
007D	0C14	470	or AL, Erase_status_val
007F	EE	471	out DX, AL ; and output the result
		472	
0080	BA2008	473	mov DX, Write_aux
0083	B0C0	474	mov AL, Nothing_en ;Ensure no write strobes
0085	EE	475	out DX, AL
0086	8A2808	476	mov DX, Write_cmd ;Set up to give some commands
0089	E8DE04	477	call Wait_for_rewind
008C	B00B	478	mov AL, Clear_board_reset ;Clear the reset condition
008E	EE	479	out DX, AL
008F	B003	480	mov AL, Set_write_status ;Set up to write
0091	EE	481	out DX, AL
0092	B001	482	mov AL, SC_cmd ;Do it!
0094	EE	483	out DX, AL

2086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 15

LCC OBJ	LINE	SOURCE
0095 EB1704	484	call Cmd_reset
	485	
0098 8A3408	486	E_wait_loop: mov DX, Read_status ;Take a look for end of tape
0099 EC	487	in AL, DX ;Get it in
009C 3408	488	xor AL, mask EOT_in ;Flip the EOT bit
009E A80A	489	test AL, mask EOT_in or mask ON_LINE_in ;Finished?
00A0 74F6	490	jz E_wait_loop
	491	
00A2 89401F	492	mov CX, Tape_settle_delay ;Now wait for tape motion to cease
00A5 E2FE	493	E_delay_loop: loop E_delay_loop
00A7 81260800F0EF	494	and Error, not New_err_bits ;Clean up the error flags
	495	
00AD 8A2808	496	mov DX, Write_cmd ;Ok, now get the board back
	497	; to normal
00B0 800A	498	mov AL, Board_reset
00B2 EE	499	out DX, AL
00B3 8002	500	mov AL, Clear_write_status ;Reset write status
00B5 EE	501	out DX, AL
	502	
00B6 EBCD04	503	call Clean_timers
	504	
00B9 1F	505	pop DS
00BA C9	506	leave
00BB C3	507	ret
	508	
	509	Erase_tape endp
	510 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 16

LSC OBJ

LINE

SOURCE

```

511 ;-----
512 ;Procedure Write_block
513 ;
514 ; FUNCTION : Writes a block to tape.
515 ;
516 ; INPUTS :
517 ;
518 ;     Write_length : Number of characters to write.
519 ;     Write_buf_addr : Start address of the write buffer.
520 ;
521 ; CALLS : Set_start, Fill, Wr_prim.
522 ;
523 ; OUTPUTS : None.
524 ;
525 ; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
526 ;
527 ;
528
529 Write_struct    struc
530
531 Write_old_BP    dw    ?
532 Write_return     dd    ?
533 Write_length     dw    ?
534 Write_buf_addr   dd    ?
535
536 Write_struct    ends
537
538 public          Write_block
539
540 Write_block      proc    far
541
542
543             enter    0, 0
544             push     DS
545             mov      AX, T_data           ;Get ready for data access
546             mov      DS, AX
547             mov      AX, [BP].Write_length ;Get the number of bytes
548             mov      Data_1, AX          ;Store it
549             les      BX, [BP].Write_buf_addr ;Get the address of the data
550             mov      AX, Start_norm      ;Get ready to set up the start
551             call     Set_start           ;Go do it
552             call     Fill                ;Go set up parity and control buffer
553             call     Wr_prim             ;Go do the write

```


8086/87/88/186 MACRO ASSEMBLER TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 17

LOC	OBJ	LINE	SOURCE
		554	
000B	1F	555	pop DS
000C	C9	556	leave
000D	CA0600	557	ret size Write_struct - 5
		558	
		559	Write_block endp
		560	
		561	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 18

LOC OBJ

LINE

SOURCE

```

562 ;-----
563 ;Procedure Write_with_long_gap
564 ;
565 ; FUNCTION : Writes a block to tape with a long inter-record gap.
566 ;
567 ; INPUTS :
568 ;
569 ;     Write_length : Number of characters to write.
570 ;     Write_buf_addr : Start address of the write buffer.
571 ;
572 ; CALLS : Set_start, Fill, Wr_prim.
573 ;
574 ; OUTPUTS : None.
575 ;
576 ; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
577 ;
578 ;
579
580 public      Write_with_long_gap
581
00E0 582 Write_with_long_gap    proc    far
583
584
585         enter    0, 0
586         push     DS
587         mov      AX, T_data          ;Get ready for data access
588         mov      DS, AX
589         mov      AX, [BP].Write_length ;Get the number of bytes
590         mov      Data_1, AX          ; and store it
591         les      BX, [BP].Write_buf_addr ;Get the address of the data
592         mov      AX, Start_long      ;Get ready to set up the start
593         call     Set_start            ;Go do it
594         call     Fill                 ;Go set up parity and control buffer
595         call     Wr_prim               ;Go do the write
596
597         pop      DS
598         leave
599         ret      size    Write_struc - 6
600
601 Write_with_long_gap    endp
602
603 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 19

LOC	OBJ	LINE	SOURCE
		604	-----
		605	;Procedure Write_EOF
		606	;
		607	; FUNCTION : Writes an EOF mark.
		608	;
		609	; INPUTS : None.
		610	;
		611	; CALLS : Set_start, Wr_prim.
		612	;
		613	; OUTPUTS : None.
		614	;
		615	; DESTROYS : AX, BX, DX, CX, SI, Flags.
		616	;
		617	;
		618	
		619	public Write_EOF
		620	
0104		621	Write_EOF proc far
		622	
		623	
0104 88000000		624	enter 0, 0
0108 1E		625	push DS
0109 B8----	R	626	mov AX, T_data ;Get ready for data access
010C 8ED8		627	mov DS, AX
010E 8EC0		628	mov ES, AX ;Also set up for Fill now
		629	
0110 B8C802		630	mov AX, Tape_mark ;Load the data to be written
0113 BE0100		631	mov SI, 1 ;Set up for loop
0116 A21020		632	mov EOF_buf[0], AL ;The first thing to be written
0119 86261300		633	mov Local_buf[0], AH ; and the parity and control
011D B90700		634	mov CX, EOF_data_gap ;The number of spaces in the
		635	; EOF mark
0120 C6441300		636	EOF_loop: mov Local_buf[SI], Nothing_en ;Do nothing in the gap
0124 46		637	inc SI
0125 E2F9		638	loop EOF_loop
0127 866413		639	mov Local_buf[SI], AH ;Trailing data is the same as
		640	; leading
012A 88841D20		641	mov EOF_buf[SI], AL
012E 46		642	inc SI
012F C6441300		643	mov Local_buf[SI], Nothing_en ;Disable
		644	
0133 C7060D0000100		645	mov Data_1, EOF_data_gap + 3 - Extra_write_bytes ;Set up for W
			r_prim

8086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 20

LOC	OBJ	LINE	SOURCE
0139	B31D20	646	mov BX, offset EOF_buf
013C	A10700	647	mov AX, Start_EOF
		648	;Set up for Set_start
013F	E30803	649	call Set_start
0142	E8C004	650	call Wr_prim
		651	;Go do it
		652	;Go do the write
0145	1F	652	pop DS
0146	C9	653	leave
0147	CB	654	ret
		655	
		656	Write_EOF endp
		657	
		658	
		659	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 21

```

LOC 08J      LINE      SOURCE
660          ;-----
661          ;Procedure Read_block
662          ;
663          ; FUNCTION : Reads a block from tape into a buffer.
664          ;
665          ; INPUTS :
666          ;     N_read_char_addr      : Address of the number of characters
667          ;     Read_buf_addr         : Start address of the read buffer
668          ;
669          ; CALLS : Clean_dma, Set_start, Cal_dma, Cmd_reset, Wait_for_end,
670          ;     Check_EOF, Cal_cc, Get_data, Clean_timers.
671          ;
672          ; OUTPUTS : Data and number of characters into the addresses above.
673          ;
674          ; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
675          ;
676          ;
677          ;
678          Read_struct      struc
679
0000          680          read_old_BP      dw      ?
0002          681          read_return     dd      ?
0006          682          N_read_char_addr dd      ?
000A          683          Read_buf_addr   dd      ?
684
685          Read_struct      ends
686
687          public          Read_block
688
0148          689          Read_block     proc    far
690
691
0148 C8000000 692          enter    0, 0
014C 1E      693          push     DS
014D B8----   R  694          mov     AX, T_data
0150 8ED8     695          mov     DS, AX
0152 EB2004   696          call    Clean_dma
0155 C45E06   697          les     BX, [BP].N_read_char_addr ;The address of the
698                                     ; number of characters
0158 268B07   699          mov     AX, word ptr ES:[BX] ; and get the number
015B A30D00   700          mov     Data_1, AX
015E C45E0A   701          les     BX, [BP].Read_buf_addr ;The address of the data
0161 050300   702          add     AX, 3 ;Space for the LRC and CRC + 1

```


8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 22

LOC	OBJ	LINE	SOURCE	
		703		; for error detection
0164	BAC3FF	704	mov DX, DMA_T0_reg	;Get ready to set up the
		705		; transfer count
0167	EF	706	out DX, AX	
0168	BA08FF	707	mov DX, DMA_T1_reg	;Get ready to set up the
		708		; transfer count
0168	EF	709	out DX, AX	
016C	B81027	710	mov AX, 10000	;Want to read 8192 chars at most
016F	BA0208	711	mov DX, Event_c	;Get ready to set up the number
		712		; of blocks
0172	EE	713	out DX, AL	
0173	BAC4	714	mov AL, AH	
0175	EE	715	out DX, AL	
0176	A10900	716	mov AX, Start_read	;Get ready to set up the start
		717		; counter
0179	E89E03	718	call Set_start	
017C	B80000	719	mov AX, 0	;High part of the IO port address
017F	BAC2FF	720	mov DX, DMA_SPH0_reg	;Get ready to set up the
		721		; source high address
0182	EF	722	out DX, AX	
0183	BA02FF	723	mov DX, DMA_SPH1_reg	;Get ready to set up the
		724		; source high address
0186	EF	725	out DX, AX	
0187	B83808	726	mov AX, Read_data	;The data port address
018A	BAC0FF	727	mov DX, DMA_SPL0_reg	;Get ready to set up the
		728		; source low address
018D	EF	729	out DX, AX	
		730		
018E	B83008	731	mov AX, Read_aux	;The data port address
0191	BA00FF	732	mov DX, DMA_SPL1_reg	;Get ready to set up the
		733		; source low address
0194	EF	734	out DX, AX	
0195	BCC0	735	mov AX, ES	;Set up to calculate the DMA
		736		; address
0197	B8F3	737	mov SI, BX	
0199	E80503	738	call Cal_DMA	;Go work them out
		739		
019C	BAC4FF	740	mov DX, DMA_DPL0_reg	;First the low part of the address
019F	EF	741	out DX, AX	
01A0	BBC1	742	mov AX, CX	;Now get the high part
01A2	BAC6FF	743	mov DX, DMA_DPH0_reg	
01A5	EF	744	out DX, AX	
		745		

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 23

LOC	CSJ	LINE	SOURCE
01A6	3CD8	746	mov AX, DS ;Now the same calculation for
		747	; the local buffer
01A8	8B1300	748	mov BX, offset Local_buf
01AB	E8F302	749	call Cal_DMA ;Go work them out
		750	
01AE	BA04FF	751	mov DX, DMA_OPL1_reg ;First the low part of the address
01B1	EF	752	out DX, AX
01B2	8BC1	753	mov AX, CX ;Now get the high part
01B4	BA06FF	754	mov DX, DMA_DPH1_reg
01B7	EF	755	out DX, AX
		756	
01B8	BACAFF	757	mov DX, DMA_CO_reg ;Now set up the DMA control registers
01BB	B846A2	758	mov AX, IO_source
01BE	EF	759	out DX, AX
01BF	BADAFF	760	mov DX, DMA_C1_reg ;Also the other register
01C2	EF	761	out DX, AX
		762	
01C3	8A1808	763	mov DX, Write_status ;Now set up the controller
01C6	A00000	764	mov AL, Out_status ;Retrieve the unit number etc
01C9	0C34	765	or AL, Read_status_val
01CB	EE	766	out DX, AL ; and output the result
01CC	8A2808	767	mov DX, Write_cmd ;Set up to give some commands
01CF	E89803	768	call Wait_for_rewind
01D2	9008	769	mov AL, Clear_board_reset ;Clear the reset condition
01D4	EE	770	out DX, AL
01D5	3001	771	mov AL, GO_cmd ;Do it!
01D7	EE	772	out DX, AL
01D8	E8D402	773	call Cmd_reset
01DB	E8D502	774	call Wait_for_end
		775	
01DE	81260800F0EF	776	and Error, not New_err_bits ;Get ready to set some new
		777	; error codes
01E4	E86E03	778	call Check_EOF ;Was there an EOF detected?
01E7	7409	779	jz Carry_on ;If Z then yes
01E9	C7060D0000000	780	mov Data_1, 0 ;No characters read
01EF	E9C000	781	jmp Read_exit
01F2	8A3008	782	mov DX, Read_aux ;Look for some errors
01F5	EC	783	in AL, DX
01F6	A810	784	test AL, Over_run_mask ;DMA error?
01F8	7509	785	jne Length_test ;If NE then not
01FA	810E080000800	786	or Error, mask DMA_err
0200	E9AF00	787	jmp Read_exit ;Fatal
		788	

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 24

LOC	OBJ	LINE	SOURCE
0203	8AC8FF	739	Length_test: mov DX, DMA_TO_reg ;This should be 1 or greater
0206	ED	790	in AX, DX
0207	29060D00	791	sub Data_1, AX ;How many were transferred?
		792	
0208	08C0	793	or AX, AX ;Set the flags
020D	750F	794	jnz Length_ok ;If NZ then no problem
020F	810E0B000400	795	Length_wrong: or Error, mask Length_err
0215	C7060D0000000	796	mov Data_1, 0
021B	E99400	797	jmp Read_exit
021E	83060D00003	798	Length_ok: add Data_1, 3 ;Correct for the extra character
0223	880E0D00	799	mov CX, Data_1 ;Get ready to convert and check
		800	; the data
0227	E3E6	801	jcxz Length_wrong
0229	8BDE	802	mov BX, SI ;Recover BX
022B	BE0000	803	mov SI, 0 ;Initialize the array index
		804	
022E	C7060F0000000	805	mov CRC, 0 ;Zero the check characters
0234	C706110000000	806	mov LRC, 0
		807	
023A	E8EF02	808	Check_loop: call Get_data ;Fetch some data and convert it
		809	
023D	8BF8	810	mov DI, AX ;Store AX for future reference
023F	80E40E	811	and AH, State_mask ;Get the state value
0242	80FC06	812	cmp AH, State_3 ;Was there a gap in front of this
0245	7419	813	je CRC? ;If EQ then yes
0247	8BC7	814	mov AX, DI ;Recover the data
0249	E8F102	815	call Cal_cc ;Go calculate the check chars
024C	0AC0	816	or AL, AL
024E	7A03	817	jpe Even_p ;Check the parity
0250	350001	818	xor AX, Parity_mask ;Now the parity bit should
		819	; always be set
0253	A90001	820	Even_p: test AX, Parity_mask ;Is it?
0256	7506	821	jnz Parity_ok
0258	810E0B000100	822	or Error, mask P_err
025E	E2DA	823	Parity_ok: loop Check_loop ;And go do the next one
		824	
		825	
0260	A10F00	826	CRC?: mov AX, CRC ;Get the CRC
0263	35D701	827	xor AX, CRC_final ;Generate the final CRC
		828	;Should a CRC have been written?
0266	25FF01	829	and AX, CH_9 ;Discard the junk
0269	741B	830	jz LRC_check
026B	FF0E0D00	831	dec Data_1 ;Correct Length for the CRC

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 25

LOC	OBJ	LINE	SOURCE
026F	49	832	dec CX ;One less character
0270	E8CA02	833	call Cal_cc ;Generate the LRC
0273	81E7FF01	834	and DI, CH_9 ;Now check the CRC
0277	3BC7	835	cmp AX, DI
0279	7406	836	je LRC_next ;If EQ then ok
027B	810E0B000200	837	or Error, mask CRC_err
		838	
0281	E8A802	839	LRC_next: call Get_data ;Get the next piece of data
0284	3BF8	840	mov DI, AX
0286	3BC7	841	LRC_check: mov AX, DI ;Get back the data
0288	8B3E1100	842	mov DI, LRC
028C	81E7FF01	843	and DI, CH_9
0290	7412	844	jz End_check ;If Z then no LRC written
		845	
0292	49	846	dec CX ;One less character
0293	FF0E0000	847	dec Data_1 ;Adjust Length for the LRC
0297	25FF01	848	and AX, CH_9 ;Get rid of the junk
029A	3BC7	849	cmp AX, DI ;Was the LRC correct
029C	7406	850	je End_check ;If EQ then ok
029E	810E0B000100	851	or Error, mask P_err
02A4	E30C	852	End_check: jcxz Read_exit ;Was that all the characters?
02A6	810E0B000400	853	or Error, mask Length_err ;If not then error
02AC	C7060D00000000	854	mov Data_1, 0
		855	
02B2	C45E06	856	Read_exit: les BX, [BP].N_read_char_addr
02B5	A10D00	857	mov AX, Data_1 ;Write the actual number of
		858	; characters
02B8	268907	859	mov word ptr ES:[BX], AX
02BB	3A2808	860	mov DX, Write_cmd ;Ok, now get the board back
		861	; to normal
02BE	800A	862	mov AL, Board_reset
02C0	EE	863	out DX, AL
02C1	B002	864	mov AL, Clear_write_status ;Reset write status
02C3	EE	865	out DX, AL
		866	
02C4	E8BF02	867	call Clean_timers
		868	
02C7	1F	869	pop DS
02C8	C9	870	leave
02C9	CA0800	871	ret size Read_struct - 6
		872	
		873	Read_block endp
		874	+1 \$EJECT

3086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 26

LOC	OBJ	LINE	SOURCE
		875	-----
		876	;Procedure Space
		877	;
		878	; FUNCTION : Spaces forward or back by a number of blocks.
		879	;
		880	; INPUTS : Space_count_addr : Address of the number of blocks to space.
		881	;
		882	; CALLS : Set_start, Wait_for_rewind, Cmd_reset, Clean_timers.
		883	;
		884	; OUTPUTS : Number of blocks spaced in Space_count_addr.
		885	;
		886	; DESTROYS : AX, BX, DX, CX, Flags.
		887	;
		888	;
		889	
----		890	Space_struct struct
		891	
0000		892	space_old_BP dw ?
0002		893	space_ret dd ?
0006		894	Space_count_addr dd ?
		895	
----		896	Space_struct ends
		897	
		898	public Space
		899	
02CC		900	Space proc far
		901	
02CC C8000000		902	enter 0, 0
02D0 1E		903	push DS
02D1 88----	R	904	mov AX, T_data
02D4 8ED8		905	mov DS, AX
02D6 BA0608		906	mov DX, Tmr_O_ctrl
02D9 B070		907	mov AL, Event_c_space_mode
02DB EE		908	out DX, AL
02DC C45E06		909	les BX, [BP].Space_count_addr ;Load the address of the
		910	; number of blocks
02DF 268B07		911	mov AX, word ptr ES:[BX] ;Get the number
02E2 A30D00		912	mov Data_1, AX ;Store it
02E5 0BC0		913	or AX, AX ;Set the flags
		914	
02E7 7503		915	jnz No_space_exit
02E9 E99400		916	jmp Space_exit ;Do nothing
02EC		917	No_space_exit:

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 27

LJC OBJ	LINE	SOURCE	
02EC B130	918	mov CL, Space_fwd_val	;Assume forward
02EE 7904	919	jns Forward	;Space reverse or forward?
	920		
02F0 F708	921	neg AX	;Get a positive count
02F2 B120	922	mov CL, Space_rev_val	;Store the output code
02F4 48	923	Forward: dec AX	;Space of one block is a
	924		; special case
02F5 750C	925	jnz Block_mode	
02F7 B81027	926	mov AX, 10000	;Max number of characters
02FA 80C904	927	or CL, Count_chars	;Set up for one block
02FD BFFF00	928	mov DI, True	;Remember this mode
0300 EB0990	929	jmp Start_space	
0303 80C908	930	Block_mode: or CL, Count_blocks	;Set up to count blocks
0306 8F0000	931	mov DI, False	;Remember this mode
0309 8BF0	932	mov SI, AX	; and remember the count
030B 8A0208	933	Start_space: mov DX, Event_c	;Get ready to set the
	934		; number of blocks
030E EE	935	out DX, AL	
030F 8AC4	936	mov AL, AH	
0311 EE	937	out DX, AL	
0312 A10900	938	mov AX, Start_read	;Get ready to set up the
	939		; start/stop values
0315 E80202	940	call Set_start	
0318 8A1808	941	mov DX, Write_status	;Get ready to set status
031B 8AC1	942	mov AL, CL	;Get the status value back
031D 0A060000	943	or AL, Out_status	;Add the unit number
0321 EE	944	out DX, AL	
0322 E84502	945	call Wait_for_rewind	
0325 BA2808	946	mov DX, Write_cmd	;Now the commands
0328 8008	947	mov AL, Clear_board_reset	
032A EE	948	out DX, AL	
032B 8001	949	mov AL, Go_cmd	
032D EE	950	out DX, AL	
032E E87E01	951	call Cmd_reset	
0331 81260B00F0EF	952	and Error, not New_err_bits	;Get ready to set some new
	953		; error codes
0337 E87901	954	call Wait_for_end	
033A E81802	955	call Check_EOF	
	956		
033D 8B160000	957	mov DX, Data_1	;Assume no need to calculate
	958		
0341 7409	959	jz No_space_EOF	
0343 0BD2	960	or DX, DX	;Set the flags

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 28

LOC	OBJ	LINE	SOURCE	
0345	7904	961	jns	One_less_space
0347	42	962	inc	DX
0348	EB0290	963	jmp	No_space_EOF
0348	4A	964	One_less_space:	dec DX
		965		
034C	89160D00	966	No_space_EOF:	mov Data_1, DX
0350	81FFFF00	967	cmp	DI, True ;Is there a need?
0354	7427	968	je	Ret_space_count ;IF E then no
		969		
0356	8A0608	970	mov	DX, Tmr_O_ctrl ;Set up to see how
		971		; many were spaced
0359	80C4	972	mov	AL, Read_back_cmd ;Set the counter for
		973		; read back mode
035B	EE	974	out	DX, AL
035C	8A0208	975	mov	DX, Event_c
035F	EC	976	in	AL, DX ;Was the counter loaded?
0360	A840	977	test	AL, mask Null_count
0362	750B	978	jnz	Not_loaded
		979		
0364	EC	980	in	AL, DX ;Get the lsb
0365	8AC8	981	mov	CL, AL ;Save it
0367	EC	982	in	AL, DX ;Get the msb
0368	8AE0	983	mov	AH, AL
036A	8AC1	984	mov	AL, CL ;Now it's in AX
036C	EB0390	985	jmp	Check_sign
		986		
036F	8BC6	987	Not_loaded:	mov AX, SI
		988		
0371	8B160D00	989	Check_sign:	mov DX, Data_1 ;Get the original
0375	0B02	990	or	DX, DX ;Set the flags
0377	7902	991	jns	Positive ;Was this forward?
0379	F7D8	992	neg	AX ;No, change sign
037B	2B00	993	Positive:	sub DX, AX ;Calculate actual value
037D		994	Ret_space_count:	
037D	268917	995	mov	word ptr ES:[CBX], DX ;Return the value
		996		
0380	3A2808	997	Space_exit:	mov DX, Write_cmd ;Ok, now get the board back
		998		; to normal
0383	800A	999	mov	AL, Board_reset
0385	EE	1000	out	DX, AL
0386	8002	1001	mov	AL, Clear_write_status ;Reset write status
0388	EE	1002	out	DX, AL
		1003		

8086/87/88/186 MACRO ASSEMBLER TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 29

LOC	OBJ	LINE	SOURCE
0389	E8FA01	1004	call Clean_timers
		1005	
038C	1F	1006	pop DS
038D	C9	1007	leave
038E	CA0400	1008	ret size Space_struct - 6
		1009	
		1010	Space
		1011	+1 \$EJECT
			endp

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 30

LOC 08J

LINE

SOURCE

```

1012 ;-----
1013 ;Procedure Cntrl_status
1014 ;
1015 ; FUNCTION : Reads the status of the controller and the tape transport.
1016 ;
1017 ; INPUTS : Status_addr : Address of the status.
1018 ;
1019 ; CALLS : None.
1020 ;
1021 ; OUTPUTS : Status in Status_addr.
1022 ;
1023 ; DESTROYS : AX, DX, Flags.
1024 ;
1025 ;
1026
----
1027 Cntrl_struct    struc
1028
0000 1029     status_old_BP    dw    ?
0002 1030     status_ret       dd    ?
0006 1031     Status_addr     dd    ?
1032
----
1033 Cntrl_struct    ends
1034
1035
1036 public         Cntrl_status
1037
0391 1038 Cntrl_status     proc    far
1039
0391 1040             enter    0,0
0395 1041             push    DS
0396 1042             mov     AX, T_data
0397 1043             mov     DS, AX
1044
039B 1045             les     BX, [BP].Status_addr    ;Load the address of the
1046                                     ; status word
039E 1047             and     Error, not New_status_bits    ;Clear the bits to be set up
03A4 1048             mov     DX, Read_status            ;Now input the info
03A7 1049             in     AL, DX
03A8 1050             or     Error, mask RWD              ;Assume rewinding
03AE 1051             test    AL, mask RWD_in
03B0 1052             jz     Online?
03B2 1053             and     Error, not mask RWD        ;Not rewinding
03B8 1054             test    AL, mask RDY_in

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 31

LOC	OBJ	LINE-	SOURCE
03BA	7406	1055	jz Online?
03BC	810E0B000001	1056	or Error, mask S_err
03C2	A802	1057	Online?: test AL, mask ON_LINE_in
03C4	7406	1058	jz LDP?
03C6	810E0B000001	1059	or Error, mask S_err
03CC	A804	1060	LDP?: test AL, mask LDP_in
03CE	7506	1061	jnz EDT?
03D0	810E0B0000020	1062	or Error, mask BOT
03D6	A808	1063	EDT?: test AL, mask EDT_in
03D8	7506	1064	jnz FPT?
03DA	810E0B001000	1065	or Error, mask EDT
03E0	A810	1066	FPT?: test AL, mask FPT_in
03E2	7506	1067	jnz End_status
03E4	810E0B000004	1068	or Error, mask HWL
03EA	268127C088	1069	End_status: and word ptr ES:[BX], not Change_bits ;Clear bits
03EF	A10800	1070	mov AX, Error
03F2	253F77	1071	and AX, Change_bits ;We only want those
03F5	260907	1072	or word ptr ES:[BX], AX
		1073	
03F8	1F	1074	pop DS
03F9	C9	1075	leave
03FA	CA0400	1076	ret size Cntrl_struct - 6
		1077	
		1078	Cntrl_status endp
		1079	
		1080	
		1081	
		1082	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 32

LOC	OBJ	LINE	SOURCE
		1083	;
		1084	;-----
		1084	;Procedure Init_tape
		1085	;
		1086	; FUNCTION : Initializes the controller.
		1087	;
		1088	; INPUTS : None.
		1089	;
		1090	; CALLS : None.
		1091	;
		1092	; OUTPUTS : None.
		1093	;
		1094	; DESTROYS : AX, DX, Flags.
		1095	;
		1096	;
		1097	
		1098	public Init_tape
		1099	
03FD		1100	Init_tape proc far
		1101	
		1102	
03FD 1E		1103	push DS
03FE B8----	R	1104	mov AX, T_data
0401 8ED8		1105	mov DS, AX
		1106	
0403 BA2008		1107	mov DX, Write_aux
0406 B000		1108	mov AL, Nothing_en ;Ensure no write strobes
0408 EE		1109	out DX, AL
		1110	
0409 BA0608		1111	mov DX, Tmr_0_cntrl ;Get ready to set up timer 0
040C B032		1112	mov AL, Start_c_init
040E EE		1113	out DX, AL ;Output the value
040F B0B2		1114	mov AL, Stop_c_init
0411 EE		1115	out DX, AL ;Output the value
0412 B072		1116	mov AL, Event_c_init
0414 EE		1117	out DX, AL ;Output the value
0415 BA0E08		1118	mov DX, Tmr_1_cntrl ;Now set up timer 1
0418 B034		1119	mov AL, Master_c_init
041A EE		1120	out DX, AL ;Output the value
041B B072		1121	mov AL, LG_det_init
041D EE		1122	out DX, AL ;Output the value
041E B0B2		1123	mov AL, SG_det_init
0420 EE		1124	out DX, AL ;Output the value
		1125	

8086/87/88/186 MACRO ASSEMBLER TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 33

LCC OBJ	LINE	SOURCE
0421 BA1808	1126	mov DX, Write_status ;Get ready to set up status
0424 B034	1127	mov AL, Read_status_val ;Set up for read, unit 0
0426 EE	1128	out DX, AL
	1129	
0427 BA2808	1130	mov DX, Write_cmd ;Get ready for commands
042A B00A	1131	mov AL, Board_reset
042C EE	1132	out DX, AL
042D B000	1133	mov AL, Clear_go_cmd
042F EE	1134	out DX, AL
0430 B002	1135	mov AL, Clear_write_status
0432 EE	1136	out DX, AL
	1137	
0433 C606000000	1138	mov Out_status, 0 ;Now initialize some variables
0438 C70608000000	1139	mov Error, 0
	1140	
043E 1F	1141	pop DS
043F CB	1142	ret
	1143	
	1144	
	1145	
	1146	Init_tape endp
	1147	
	1148 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 34

LOC 08J

LINE

SOURCE

```

1149 ;-----
1150 ;Procedure Diag
1151 ;
1152 ; FUNCTION : Obtains diagnostics information from stored data.
1153 ;
1154 ; INPUTS :
1155 ;
1156 ;     State_addr      : Address of the state of the controller at the time
1157 ;                     that the selected data was read.
1158 ;     Par_err_addr    : Address of the correct/incorrect parity flag
1159 ;     Index           : Index into the data buffer
1160 ;     Diag_buf_addr   : Start address of the data buffer
1161 ;
1162 ; CALLS : None.
1163 ;
1164 ; OUTPUTS : State and error flag into the above addresses
1165 ;
1166 ; DESTROYS : AX, BX, SI, ES, Flags.
1167 ;
1168 ;
1169
----
1170 Diag_struct    struc
1171
0000 1172 diag_old_BP    dw      ?
0002 1173 diag_return     dd      ?
0006 1174 State_addr     dd      ?
000A 1175 Par_err_addr    dd      ?
000E 1176 Index         dw      ?
0010 1177 Diag_buf_addr  dd      ?
1178
----
1179 Diag_struct    ends
1180
1181
1182 public        Diag
1183
0440 1184 Diag           proc    far
1185
1186
0440 C8000000 1187             enter    0, 0
0444 1E        1188             push     DS
0445 B8----- R 1189             mov      AX, T_data
0448 8ED8      1190             mov      DS, AX
1191

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 35

LDC OBJ	LINE	SOURCE
044A 8B760E	1192	mov SI, [BP].Index ;Get the index into the data
044D C45E10	1193	les BX, [BP].Diag_buf_addr ;Get the buffer address
	1194	
0450 268A00	1195	mov AL, byte ptr ES:[BX][SI];Now to check parity
0453 8A6413	1196	mov AH, Local_buf[SI]
0456 C45E0A	1197	les BX, [BP].Par_err_addr ;Where the error info goes
0459 26C60700	1198	mov byte ptr ES:[BX], False ;Assume no error
045D 0AC0	1199	or AL, AL ;Set the flags
	1200	
045F 7B03	1201	jpo Diag_odd_p
0461 350001	1202	xor AX, Parity_mask ;Now the parity bit should
	1203	; be 0
0464 A90001	1204	Diag_odd_p: test AX, Parity_mask ;Is it?
0467 7404	1205	jz Diag_p_ok ;If Z then ok
0469 26C607FF	1206	mov byte ptr ES:[BX], True ;Signal parity error
	1207	
046D C45E06	1208	Diag_p_ok: les BX, [BP].State_addr ;Where the state value goes
0470 8AC4	1209	mov AL, AH ;Get the state info to AL
0472 250E00	1210	and AX, State_mask ;Get rid of any junk
0475 01F8	1211	sar AX, 1 ;Get the state into the
	1212	; lowest three bits
0477 268907	1213	mov word ptr ES:[BX], AX ;Store the state value
	1214	
047A 1F	1215	pop DS
047B C9	1216	leave ;Ok, that's it
047C CA0E00	1217	ret size Diag_struct - 6
	1218	
	1219	Diag endp
	1220	
	1221	+1 \$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 36

LOC OBJ

LINE

SOURCE

```

1222 ;-----
1223 ;Procedure Off_line
1224 ;
1225 ; FUNCTION : Sets the selected unit off-line.
1226 ;
1227 ; INPUTS : None.
1228 ;
1229 ; CALLS : Cmd_reset.
1230 ;
1231 ; OUTPUTS : None.
1232 ;
1233 ; DESTROYS : AX, DX, Flags.
1234 ;
1235 ;
1236
1237 public      Off_line
1238
1239 Off_line    proc    far
1240
1241
1242          mov     DX, Write_cmd      ;Get ready to give the command
1243          mov     AL, Off_line_cmd
1244          out     DX, AL             ; and give it
1245          call    Cmd_reset          ;Now reset that line
1246
1247          ret                               ;Thats it!
1248
1249 Off_line    endp
1250 +1 $EJECT

```

047F

047F 8A2808

0482 B007

0484 EE

0485 E82700

0488 CB

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 37

```

LOC 08J      LINE      SOURCE
              1251      ;-----
              1252      ;Procedure Rewind_tape
              1253      ;
              1254      ; FUNCTION : Rewinds the selected tape.
              1255      ;
              1256      ; INPUTS : None.
              1257      ;
              1258      ; CALLS : Cmd_reset.
              1259      ;
              1260      ; OUTPUTS : None.
              1261      ;
              1262      ; DESTROYS : AX, DX, Flags.
              1263      ;
              1264      ;
              1265
              1266      public      Rewind_tape
0489          1267
              1268      Rewind_tape  proc    far
              1269
              1270
0489 BA2808    1271                  mov     DX, Write_cmd      ;Get ready to give the command
048C B005      1272                  mov     AL, Rewind_cmd
048E EE        1273                  out     DX, AL              ; and give it
048F E81D00    1274                  call    Cmd_reset          ;Now reset that line
              1275
0492 CB        1276                  ret                          ;Thats it!
              1277
              1278      Rewind_tape  endp
              1279
              1280 +1  $EJECT

```

3086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 38

LOC	OBJ	LINE	SOURCE
		1281	-----
		1282	;Procedure Set_RTH_high
		1283	;
		1284	; FUNCTION : Sets the read threshold high.
		1285	;
		1286	; INPUTS : None.
		1287	;
		1288	; CALLS : None.
		1289	;
		1290	; OUTPUTS : None.
		1291	;
		1292	; DESTROYS : AX, DX, Flags.
		1293	;
		1294	;
		1295	
		1296	public Set_RTH_high
		1297	
0493		1298	Set_RTH_high proc far
		1299	
		1300	
0493 3A2808		1301	mov DX, Write_cmd ;Get ready to give the command
0496 8009		1302	mov AL, RTH_high_cmd
0498 EE		1303	out DX, AL ; and give it
		1304	
0499 CB		1305	ret ;Thats it!
		1306	
		1307	Set_RTH_high endp
		1308	;\$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 39

LOC	OBJ	LINE	SOURCE
		1309	;
		1310	;-----
		1310	;Procedure Set_RTH_low
		1311	;
		1312	; FUNCTION : Sets the read threshold low.
		1313	;
		1314	; INPUTS : None.
		1315	;
		1316	; CALLS : None.
		1317	;
		1318	; OUTPUTS : None.
		1319	;
		1320	; DESTROYS : AX, DX, Flags.
		1321	;
		1322	;
		1323	
		1324	public Set_RTH_low
		1325	
049A		1326	Set_RTH_low proc far
		1327	
		1328	
049A BA2808		1329	mov DX, Write_cmd ;Get ready to give the command
049D B008		1330	mov AL, RTH_low_cmd
049F EE		1331	out DX, AL ; and give it
		1332	
04A0 C8		1333	ret ;Thats it!
		1334	
		1335	Set_RTH_low endp
		1336 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 40

LOC OBJ

LINE

SOURCE

```

1337 ;*****
1338 ;*****
1339 ;
1340 ; LOCAL PROCEDURES
1341 ;
1342 ;*****
1343 ;*****
1344 ;-----
1345 ;Procedure Cal_dma
1346 ;
1347 ; FUNCTION : Calculates the values to write to the DMA registers. The input
1348 ;             address is converted from segment/offset form to 20 bit form.
1349 ;
1350 ; INPUTS : AX:BX : Input address.
1351 ;
1352 ; CALLS : None.
1353 ;
1354 ; OUTPUTS : CX:8X : Output address.
1355 ;
1356 ; DESTROYS : AX, Flags.
1357 ;
1358 ;
1359
04A1 1360 Cal_dma      proc      near
1361
04A1 C1C004 1362          rol      AX, 4          ;Now the highest 4 bits are
1363                                     ; in bits 0 to 3
04A4 8BC8    1364          mov     CX, AX          ;Store the segment in CX
04A6 25F0FF 1365          and     AX, 0fff0h      ;Get the part of the segment to
1366                                     ; be added
04A9 03C3    1367          add     AX, BX          ;Add it
04AB 7301    1368          jnc     No_carry        ;If there was a carry we must
1369                                     ; add 1 to the high part
04AD 41      1370          inc     CX
04AE C3      1371 No_carry:   ret
1372
1373 Cal_dma      endp
1374
1375 +1 $EJECT

```


8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 41

```

LOC OBJ      LINE      SOURCE
1376          ;-----
1377          ;Procedure Cmd_reset
1378          ;
1379          ; FUNCTION : Resets the command bit specified in AX.
1380          ;
1381          ; INPUTS : AX : Command bit address and value.
1382          ;          DX : Address of the command port.
1383          ;
1384          ; CALLS : None.
1385          ;
1386          ; OUTPUTS : None.
1387          ;
1388          ; DESTROYS : AX, Flags.
1389          ;
1390          ;
1391          Cmd_reset      proc      near
1392
1393
1394          xor            AL, 1          ;Flip bit 0 (the data bit)
1395          out            DX, AL        ; and output it
1396          ret
1397
1398          Cmd_reset      endp
1399
1400 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 42

LOC	OBJ	LINE	SOURCE
		1401	;
		1402	;Procedure Wait_for_end
		1403	;
		1404	; FUNCTION : Waits for a command cycle to terminate. This may be as a result
		1405	; normal command termination, time_out, the tape unit going off_line,
		1406	; or encountering the BOT tab backwards.
		1407	;
		1408	; INPUTS : None.
		1409	;
		1410	; CALLS : None.
		1411	;
		1412	; OUTPUTS : None.
		1413	;
		1414	; DESTROYS : AX, DX, CX, Flags.
		1415	;
		1416	;
		1417	;
0483		1418	Wait_for_end proc near
		1419	
		1420	
0483 893200		1421	mov CX, Run_delay ;Wait for the controller to go
0486 E2FE		1422	Run_loop: loop Run_loop
0488 BA3408		1423	mov DX, Read_status ;Get ready to input the RUN bit
048B 810E0B000020		1424	or Error, mask BOT
04C1 89FFFF		1425	mov CX, OFFFh ;Set initial time out
04C4 BA3408		1426	Wait_loop: mov DX, Read_status ;Take a look at RUN and ON_LINE
04C7 EC		1427	in AL, DX ;Get it in
04C8 354000		1428	xor AX, mask RUN ;Invert the run bit
04CB A842		1429	test AL, mask RUN or mask ON_LINE_in ;Finished?
04CD 7534		1430	jnz End_det
		1431	
04CF A804		1432	test AL, mask LDP_in ;Are we at BOT?
04D1 7511		1433	jnz Not_LDP
04D3 F7060B000020		1434	test Error, mask BOT ;Were we at BOT?
04D9 7428		1435	jz End_det ;If not then end of op
		1436	
04DB 810E0B000020		1437	or Error, mask BOT ;Set up for next time
		1438	
04E1 EB0790		1439	jmp Gap_check
04E4 81260B00FFDF		1440	Not_LDP: and Error, not mask BOT ;Not at BOT
		1441	
04EA BA0E08		1442	Gap_check: mov DX, Tar_1_cntrl ;Now check for any tape activity
04ED 80E4		1443	mov AL, Read_back_L6 ;Set up to read the counter output

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 43

LOC	OBJ	LINE	SOURCE
04EF	EE	1444	out DX, AL
		1445	
04F0	8A0A08	1446	mov DX, LG_det ;Get the counter status
04F3	EC	1447	in AL, DX
04F4	A880	1448	test AL, mask Output
04F6	7503	1449	jnz No_gap ;If output is one then no gap
04F8	89FFFF	1450	mov CX, 0FFFFh ;If a gap then reset the time out
04FB	E2C7	1451	No_gap: loop Wait_loop
		1452	
04FD	810E08000010	1453	or Error, mask Time_err ;If we get here then time out
		1454	
0503	BACAFF	1455	End_det: mov DX, DMA_CO_reg ;Now stop the DMA
0506	B8B416	1456	mov AX, Stop_DMA
0509	EF	1457	out DX, AX
050A	BADAFF	1458	mov DX, DMA_C1_reg ;Also the other register
050D	EF	1459	out DX, AX
		1460	
050E	8A2008	1461	mov DX, Write_aux
0511	B000	1462	mov AL, Nothing_en ;Ensure no write strobes
0513	EE	1463	out DX, AL
		1464	
0514	B9401F	1465	mov CX, Tape_settle_delay ;Now wait for tape motion to cease
0517	E2FE	1466	Delay_loop: loop Delay_loop
		1467	
0519	C3	1468	ret
		1469	
		1470	Wait_for_end endp
		1471 +1	\$EJECT

3086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 44

LOC 02J

LINE

SOURCE

```

1472 ;-----
1473 ;Procedure Set_start
1474 ;
1475 ; FUNCTION : The values of the start and stop counters are set up.
1476 ;
1477 ; INPUTS : AX : Start count value.
1478 ;
1479 ; CALLS : None.
1480 ;
1481 ; OUTPUTS : None.
1482 ;
1483 ; DESTROYS : AX, DX, Flags.
1484 ;
1485 ;
1486
051A 1487 Set_start      proc      near
1488
051A 1489             mov     DX, Start_c      ;Start counter address
051D 1490             out     DX, AL
051E 1491             mov     AL, AH
0520 1492             out     DX, AL
0521 1493             mov     DX, Stop_c      ;Now the stop counter
0524 1494             mov     AX, Stop_norm ;Always the same
0527 1495             out     DX, AL
0528 1496             mov     AL, AH
052A 1497             out     DX, AL
1498
052B 1499             ret
1500
1501 Set_start      endp
1502
1503 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 45

```

LOC OBJ          LINE    SOURCE
1504              ;-----
1505              ;Procedure Get_data
1506              ;
1507              ; FUNCTION : Gets a data byte as well as its parity and state from the data
1508              ;                buffer and local buffer respectively.
1509              ;
1510              ; INPUTS :
1511              ;     ES:BX   : Start address of the data buffer.
1512              ;     SI     : Index into buffer.
1513              ;
1514              ; CALLS : None.
1515              ;
1516              ; OUTPUTS :
1517              ;     AX     : Data, parity in bit 8, state in bits 9-11.
1518              ;     SI     : Index incremented.
1519              ;
1520              ; DESTROYS : Flags.
1521              ;
1522              ;
1523
052C              1524      Get_data      proc      near
1525
052C 8A6413         1526              mov     AH, Local_buf[SI]      ;Get the parity etc
052F 268A00         1527              mov     AL, byte ptr ES:[BX][SI]      ; and the data
0532 35FF01         1528              xor     AX, Invert_mask      ;Invert the data and parity
0535 886413         1529              mov     Local_buf[SI], AH      ;Now put back corrected data
0538 268800         1530              mov     byte ptr ES:[BX][SI], AL
053B 46             1531              inc     SI              ;For next time
1532
053C C3            1533              ret
1534
1535
1536      Get_data      endp
1537 +1 $EJECT

```


8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 46

LOC OBJ

LINE

SOURCE

```

1538 ;-----
1539 ;Procedure Cal_cc
1540 ;
1541 ; FUNCTION : Calculates the new CRC.
1542 ;
1543 ; INPUTS : Character and parity in AX.
1544 ;
1545 ; CALLS : None.
1546 ;
1547 ; OUTPUTS : None.
1548 ;
1549 ; DESTROYS : Flags.
1550 ;
1551 ;
1552
053D 1553 Cal_cc      proc      near
1554
053D 50 1555          push    AX          ;Save AX
053E 31061100 1556          xor     LRC, AX      ;Generate the LRC
1557
0542 33060F00 1558          xor     AX, CRC        ;The first step
0546 25FF01 1559          and     AX, CH_9
0549 0108 1560          rcr     AX, 1          ;Is the bit to be fed back
1561          ; a 1?
0548 7303 1562          jnc     No_feedback    ;If 0 then the same as doing
1563          ; nothing
0540 353C01 1564          xor     AX, CRC_feedback ;If 1 then xor the bits which
1565          ; are fed back
0550 A30F00 1566 No_feedback: mov     CRC, AX    ;Store it
1567
0553 58 1568          pop     AX          ;Restore AX
1569
0554 C3 1570          ret
1571
1572 Cal_cc      endp
1573
1574 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 47

```

LOC OBJ          LINE    SOURCE
                  1575    ;-----
                  1576    ;Procedure Check_EOF
                  1577    ;
                  1578    ; FUNCTION : Checks for the detection of a tape mark after read operations.
                  1579    ;
                  1580    ; INPUTS : None.
                  1581    ;
                  1582    ; CALLS : None.
                  1583    ;
                  1584    ; OUTPUTS : Z flag set if not detected.
                  1585    ;
                  1586    ; DESTROYS : AX, DX, Flags.
                  1587    ;
                  1588    ;
                  1589
0555             1590    Check_EOF      proc      near
                  1591
0555 8A3408       1592                mov     DX, Read_status    ;Get ready to input the status byte
0558 81260800FFFF 1593                and     Error, not mask EOF  ;Clear the EOF bit
055E EC          1594                in      AL, DX
055F A880        1595                test   AL, mask EOF_in      ;Did we encounter an EOF
0561 7406        1596                jz      Check_EOF_exit
0563 810E08002000 1597                or      Error, mask EOF
                  1598
0569 C3         1599    Check_EOF_exit: ret
                  1600
                  1601    Check_EOF      endp
                  1602
                  1603 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 48

LOC	OBJ	LINE	SOURCE
		1604	;-----
		1605	;Procedure Wait_for_rewind
		1606	;
		1607	; FUNCTION : Waits for a rewind operation to complete before returning.
		1608	;
		1609	; INPUTS : None.
		1610	;
		1611	; CALLS : None.
		1612	;
		1613	; OUTPUTS : None.
		1614	;
		1615	; DESTROYS : AX, Flags.
		1616	;
		1617	;
		1618	
056A		1619	Wait_for_rewind proc near
		1620	
056A 52		1621	push DX ;Save DX
056B BA3408		1622	mov DX, Read_status ;Get ready to input RWD
056E EC		1623	Rewinding: in AL, DX
056F A820		1624	test AL, mask RWD_in ;Is it rewinding?
0571 74FB		1625	jz Rewinding ;If Z then yes
0573 5A		1626	pop DX
		1627	
0574 C3		1628	ret
		1629	
		1630	
		1631	
		1632	
		1633	Wait_for_rewind endp
		1634	
		1635 +1	\$EJECT

8086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 49

LDC OBJ

LINE

SOURCE

```

1636 ;-----
1637 ;Procedure Clean_dma
1638 ;
1639 ; FUNCTION : Resets all DMA request lines on the controller cards.
1640 ;
1641 ; INPUTS : None.
1642 ;
1643 ; CALLS : None.
1644 ;
1645 ; OUTPUTS : None.
1646 ;
1647 ; DESTROYS : AX, DX, Flags.
1648 ;
1649 ;
1650
0575 1651 Clean_dma      proc      near
1652
0575 1653             mov     DX, Write_data
0578 EE 1654             out     DX, AL
0579 BA2008 1655             mov     DX, Write_aux
057C EE 1656             out     DX, AL
057D BA3808 1657             mov     DX, Read_data
0580 EC 1658             in      AL, DX
0581 BA3008 1659             mov     DX, Read_aux
0584 EC 1660             in      AL, DX
1661
0585 C3 1662             ret
1663
1664 Clean_dma      endp
1665 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 50

```

LOC OBJ          LINE    SOURCE
1666             ;-----
1667             ;Procedure Clean_timers
1668             ;
1669             ; FUNCTION : Resets all 8254 timers. This must be done as the 8254s do not
1670             ;             operate in the same way when coming out of reset as when
1671             ;             having completed a timing cycle.
1672             ;
1673             ; INPUTS : None.
1674             ;
1675             ; CALLS : None.
1676             ;
1677             ; OUTPUTS : None.
1678             ;
1679             ; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
1680             ;
1681             ;
1682             ;
0586             1683      Clean_timers      proc      near
1684
0586 BA0608        1685                  mov     DX, Tmr_0_ctrl      ;Now clear the counters
0589 B032         1686                  mov     AL, Start_c_init
0588 EE          1687                  out     DX, AL              ;Output the value
058C B082        1688                  mov     AL, Stop_c_init
058E EE          1689                  out     DX, AL              ;Output the value
058F B072        1690                  mov     AL, Event_c_init
0591 EE          1691                  out     DX, AL              ;Output the value
1692
0592 C3          1693                  ret
1694
1695      Clean_timers      endp
1696 +1 $EJECT

```

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 51

LOC	OBJ	LINE	SOURCE
		1697	-----
		1698	;Procedure Fill
		1699	;
		1700	; FUNCTION : Generates parity, strobe enable information and the CRC for a
		1701	; block to be written to tape.
		1702	;
		1703	; INPUTS : ES:BX : Data buffer start address.
		1704	;
		1705	; CALLS : Cal_cc.
		1706	;
		1707	; OUTPUTS : None.
		1708	;
		1709	; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
		1710	;
		1711	;
0593		1712	Fill proc near
		1713	
0593 8B0E0000		1714	mov CX, Data_L ;Load the data length
0597 BE0000		1715	mov SI, 0 ;Zero the index
059A C7060F000000		1716	mov CRC, 0 ; and the CRC
		1717	
05A0 E354		1718	jcxz No_wr_data
05A2 268A00		1719	Fill_loop: mov AL, byte ptr ES:[BX][SI] ;Get a byte
05A5 0AC0		1720	or AL, AL ;Set the flags
05A7 B401		1721	mov AH, 1 ;Assume even parity
05A9 7A02		1722	jpe Even_parity ;Is it?
05AB 3400		1723	mov AH, 0 ;No
05AD 80CC02		1724	Even_parity: or AH, WDS_en ;We want to generate a WDS
		1725	; strobe
05B0 8B6413		1726	mov Local_buf[SI], AH ;Now store it
05B3 E887FF		1727	call Cal_cc ;Generate the CRC
05B6 46		1728	inc SI ;Next byte.....
05B7 E2E9		1729	loop Fill_loop ;Finished with data?
		1730	
05B9 8000		1731	mov AL, Nothing_en ;Now we have 3 spaces
05BB 8B4413		1732	mov Local_buf[SI], AL ;Store into the buffer
05BE 46		1733	inc SI ;Next location
05BF 8B4413		1734	mov Local_buf[SI], AL ;Store into the buffer
05C2 46		1735	inc SI ;Next location
05C3 8B4413		1736	mov Local_buf[SI], AL ;Store into the buffer
05C6 46		1737	inc SI ;Next location
		1738	
05C7 A10F00		1739	mov AX, CRC ;Now for the CRC

3086/87/88/186 MACRO ASSEMBLER

TAPE_IO_CONTROL_MODULE

16/06/86 PAGE 52

LOC	OBJ	LINE	SOURCE
05CA	35D701	1740	xor AX, CRC_final ;Generate the final CRC
05CD	25FF01	1741	and AX, CH_9 ;Get rid of the junk
05D0	268800	1742	mov byte ptr ES:[BX][SI], AL ;Now store the data
05D3	80CC02	1743	or AH, WDS_en ;We want WDS strobe
05D6	886413	1744	mov Local_buf[SI], AH ;Store the parity
05D9	46	1745	inc SI ;Next location
		1746	
05DA	3000	1747	mov AL, Nothing_en ;And three more spaces
05DC	884413	1748	mov Local_buf[SI], AL ;Store into the buffer
05DF	46	1749	inc SI ;Next location
05E0	884413	1750	mov Local_buf[SI], AL ;Store into the buffer
05E3	46	1751	inc SI ;Next location
05E4	884413	1752	mov Local_buf[SI], AL ;Store into the buffer
05E7	46	1753	inc SI ;Next location
		1754	
05E8	8004	1755	mov AL, WARS_en ;Now generate a WARS strobe
05EA	884413	1756	mov Local_buf[SI], AL ;Store into the buffer
05ED	46	1757	inc SI
		1758	
05EE	8000	1759	mov AL, Nothing_en ;Disable all
05F0	884413	1760	mov Local_buf[SI], AL
05F3	EB0F90	1761	jmp Fill_exit
		1762	
05F6	390900	1763	No_wr_data: mov CX, Extra_write_bytes
05F9	8000	1764	mov AL, Nothing_en
05FB	8E0000	1765	mov SI, 0
05FE	884413	1766	No_wr_loop: mov Local_buf[SI], AL
0601	46	1767	inc SI
0602	E2FA	1768	loop No_wr_loop
		1769	
		1770	
0604	C3	1771	Fill_exit: ret ;Ok, thats it
		1772	
		1773	Fill endp
		1774	
		1775	+1 \$EJECT

3086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 53

LDC OBJ

LINE

SOURCE

```

1776 ;-----
1777 ;Procedure Wr_pria
1778 ;
1779 ; FUNCTION : Executes a write operation on a block of data with parity and
1780 ;           strobe information already set up.
1781 ;
1782 ; INPUTS : None.
1783 ;
1784 ; CALLS : Cal_dma, Wait_for_rewind, Cmd_reset, Wait_for_end, Clean_timers,
1785 ;         Clean_dma.
1786 ;
1787 ; OUTPUTS : None.
1788 ;
1789 ; DESTROYS : AX, BX, DX, CX, SI, DI, ES, Flags.
1790 ;
1791 ;

```

```

0605 1792
1793 Wr_pria      proc      near
1794
1795
0605 E860FF 1796      call    Clean_dma
0608 A10D00 1797      mov     AX, Data_l      ;Get the data length
060B 050900 1798      add     AX, Extra_write_bytes ;Adjust for the extra space
1799
060E BAC8FF 1800      mov     DX, DMA_T0_reg   ;Get ready to set up the
1801                                ; transfer count
0611 EF 1802      out     DX, AX
0612 BAD8FF 1803      mov     DX, DMA_T1_reg   ;Get ready to set up the
1804                                ; transfer count
0615 EF 1805      out     DX, AX
0616 BA0208 1806      mov     DX, Event_c     ;Get ready to set up the number
1807                                ; of characters
0619 EE 1808      out     DX, AL
061A BAC4 1809      mov     AL, AH
061C EE 1810      out     DX, AL
1811
061D 880000 1812      mov     AX, 0           ;High part of the IO port address
0620 BAC6FF 1813      mov     DX, DMA_DPH0_reg ;Get ready to set up the
1814                                ; destination high address
0623 EF 1815      out     DX, AX
0624 BAD6FF 1816      mov     DX, DMA_DPH1_reg ;Get ready to set up the
1817                                ; destination high address
0627 EF 1818      out     DX, AX

```


8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 54

LOC	OBJ	LINE	SOURCE
0628	B81008	1819	mov AX, Write_data ;The data port address
062B	BAC4FF	1820	mov DX, DMA_DPLO_reg ;Get ready to set up the
		1821	; destination low address
062E	EF	1822	out DX, AX
		1823	
062F	B82008	1824	mov AX, Write_aux ;The data port address
0632	BAD4FF	1825	mov DX, DMA_DPL1_reg ;Get ready to set up the
		1826	; destination low address
0635	EF	1827	out DX, AX
0636	BCC0	1828	mov AX, ES ;Set up to calculate the DMA
		1829	; address
0638	B8F3	1830	mov SI, BX
063A	E864FE	1831	call Cal_DMA ;Go work them out
		1832	
063D	BAC0FF	1833	mov DX, DMA_SPLO_reg ;First the low part of the address
0640	EF	1834	out DX, AX
0641	B8C1	1835	mov AX, CX ;Now get the high part
0643	BAC2FF	1836	mov DX, DMA_SPH0_reg
0646	EF	1837	out DX, AX
		1838	
0647	B8D8	1839	mov AX, DS ;Now the same calculation for
		1840	; the local buffer
0649	B81300	1841	mov BX, offset Local_buf
064C	E852FE	1842	call Cal_DMA ;Go work them out
		1843	
064F	BAD0FF	1844	mov DX, DMA_SPL1_reg ;First the low part of the address
0652	EF	1845	out DX, AX
0653	B8C1	1846	mov AX, CX ;Now get the high part
0655	BAD2FF	1847	mov DX, DMA_SPH1_reg
0658	EF	1848	out DX, AX
		1849	
0659	BACAFF	1850	mov DX, DMA_CO_reg ;Now set up the DMA control registers
065C	B88616	1851	mov AX, ID_dest
065F	EF	1852	out DX, AX
0660	BADAFF	1853	mov DX, DMA_C1_reg ;Also the other register
0663	EF	1854	out DX, AX
		1855	
0664	BA1808	1856	mov DX, Write_status ;Now set up the controller
0667	A00000	1857	mov AL, Out_status ;Retrieve the unit number etc
066A	0C10	1858	or AL, Write_status_val
066C	EE	1859	out DX, AL ; and output the result
		1860	
066D	BA2008	1861	mov DX, Write_aux

8086/87/88/186 MACRO ASSEMBLER

TAPE_ID_CONTROL_MODULE

16/06/86 PAGE 55

LOC	OBJ	LINE	SOURCE
0670	3000	1862	mov AL, Nothing_en ;Ensure no write strobes
0672	EE	1863	out DX, AL
0673	BA2808	1864	mov DX, Write_cmd ;Set up to give some commands
0676	E8F1FE	1865	call Wait_for_rewind
0679	B008	1866	mov AL, Clear_board_reset ;Clear the reset condition
067B	EE	1867	out DX, AL
067C	B003	1868	mov AL, Set_write_status ;Set up to write
067E	EE	1869	out DX, AL
067F	B001	1870	mov AL, GO_cmd ;Do it!
0681	EE	1871	out DX, AL
0682	E82AFE	1872	call Cmd_reset
0685	E828FE	1873	call Wait_for_end
		1874	
0688	81260B00F0EF	1875	and Error, not New_err_bits ;Get ready to set some new
		1876	; error codes
		1877	
068E	BA3008	1878	mov DX, Read_aux ;Take a look at the controller
0691	EC	1879	in AL, DX
0692	A810	1880	test AL, Over_run_mask ;Was there a DMA error?
0694	7506	1881	jne Not_write_err
0696	810E0B000800	1882	or Error, mask DMA_err
		1883	
069C	81260B000FFF	1884	Not_write_err: and Error, not mask EOF ;Clear the EOF bit
06A2	BA2808	1885	mov DX, Write_cmd ;Ok, now get the board back
		1886	; to normal
06A5	B00A	1887	mov AL, Board_reset
06A7	EE	1888	out DX, AL
06A8	B002	1889	mov AL, Clear_write_status ;Reset write status
06AA	EE	1890	out DX, AL
		1891	
06AB	E8D8FE	1892	call Clean_timers
		1893	
		1894	
06AE	C3	1895	ret ;That's it
		1896	
		1897	
		1898	Wr_prim endp
		1899	
		1900	
----		1901	T_code ends
		1902	end

ASSEMBLY COMPLETE, NO ERRORS FOUND

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 1

SERIES-III 8086/87/88/186 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE COMMS_MODULE

OBJECT MODULE PLACED IN :F3:ADMCOM.08J

ASSEMBLER INVOKED BY: ASM86.86 :F3:ADMCOM.ASM

LOC	OBJ	LINE	SOURCE
		1 +1	\$DEBUG
		2 +1	\$GENONLY
		3 +1	\$TYPE
		4 +1	\$PAGELENGTH(48)
		5 +1	\$MOD186
		6	;
		7	;*****
		8	;*****
		9	;
		10	; MODULE FUNCTION :
		11	;
		12	;This module is the data link between the tape controller system and the
		13	; host computer. Communication is via a frame structured protocol.
		14	; NOTE : This module also contains the procedure to set the terminal baud
		15	; rate.
		16	;
		17	;*****
		18	;
		19	; Written : A.D.McGuffog 7 October 1985.
		20	;
		21	;*****
		22	;
		23	; Language : Intel ASM 86.
		24	;
		25	;*****
		26	;
		27	; Other Software Required : Designed to run with Intel Pascal-86 V3
		28	;
		29	;*****
		30	;
		31	; Hardware required : 80188 CPU card with an 8251A installed as USART0.
		32	; Serial link starts at 9600 baud 8 bits , no parity.
		33	;
		34	;*****
		35	;
		36 +1	; Restrictions : Baud rate generation is accurate only to 2.4% at
		37	; 9600 baud.
		38	;

5086/67/88/156 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 2

LOC C87

LINE

SOURCE

```

39 ;*****
40 ;
41 ;           Module description :
42 ;
43 ;This module provides the data link between the controller and the host
44 ; computer. This data link is a serial RS232 line, normally operating at
45 ; 9600 baud. This module is controlled by the next layer of software via
46 ; Pascal compatible procedure calls. Procedures are provided to do the
47 ; following :
48 ;
49 ;     1. Initialize the data link.
50 ;
51 ;     2. Set the baud rates of both the data link and the terminal.
52 ;
53 ;     3. Transmit a frame.
54 ;
55 ;     4. Receive a frame.
56 ;
57 ;     5. Set up a timeout.
58 ;
59 ; All communication from this module to higher levels are done via software
60 ; interrupts. Note that these transfer no data. Data is transferred when the
61 ; next software layer executes a procedure call in response to the interrupt.
62 ; The following interrupts are used :
63 ;
64 ;     1. Frame received interrupt.(int 100)
65 ;
66 ;     2. Timeout interrupt.(int 101)
67 ;
68 ; The software is driven by three hardware interrupts :
69 ;
70 ;     1. Received character(int 13).
71 ;
72 ;     2. Transmit buffer empty(int 12).
73 ;
74 ;     3. Timer tick(int 19).
75 ;
76 ; Data protocol : The link makes use of a frame protocol of the PAR-(Positive
77 ; Acknowledgement with Retransmission) type. All frames have CRC's for error
78 ; detection.
79 ;
80 ; Frame Formats : Three valid frames exist, two of which carry data, and two
81 ; acknowledgement information. These are :

```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 3

LCC DBJ

LINE SOURCE

```
82 :  
83 :  
84 : 1. Data frames. These contain data, and the terminating character  
85 : indicates whether further data is to be sent, e.i. further frames  
86 : are still to be received before the command block is complete.  
87 :  
88 : 2. Ack frames. Acknowledgement of the frame indicated by the frame  
89 : number field.  
90 :  
91 : 3. Nak frames. As for Ack frames only error in the indicated frame.  
92 : The format for all frames is the same :  
93 :  
94 : DLE  
95 : First character (DATA, ACK, NAK)  
96 : Frame number (0, 1)  
97 : Data ( for start and end frames only)  
98 : .  
99 : .  
100 : .  
101 : DLE  
102 : Terminating character (END, CONT)  
103 : CRC (low byte)  
104 : CRC (high byte)  
105 :  
106 : NOTE : Data may be of any length, and makes use of character stuffing of  
107 : DLE characters.  
108 :  
109 : The CRC is CCITT-16 standard.  
110 :  
111 :  
112 :*****  
113 :*****  
114 +1 $EJECT
```

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 4

LOC OBJ

LINE

SOURCE

```

115 ;
116 +1 $INCLUDE(:"F3:A0MDEF.ASM)
=1 117 ;
=1 118 ;
=1 119 ;
=1 120 ;
=1 121 ;
=1 122 ;
=1 123 ;
C MACRO =1 124 codemacro reset
# =1 125 db 0FAh
# =1 126 db 0EAh
# =1 127 dw 0h
# =1 128 dw 0FFFFh
# =1 129 endm
=1 130
=1 131 ; Now the equates for the on_chip timers :
=1 132
FF00 =1 133 Per_base equ 0FF00h ;Base address of the 80188
=1 134 ; peripheral control block
FF50 =1 135 CNT0_reg equ Per_base + 050h ;The three count registers :
FF52 =1 136 MCNT0A_reg equ Per_base + 052h
FF54 =1 137 MCNT0B_reg equ Per_base + 054h
FF56 =1 138 TMROCNTR_reg equ Per_base + 056h
FF58 =1 139 CNT1_reg equ Per_base + 058h
FF5A =1 140 MCNT1A_reg equ Per_base + 05Ah
FF5C =1 141 MCNT1B_reg equ Per_base + 05Ch
FF5E =1 142 TMR1CNTR_reg equ Per_base + 05Eh
FF60 =1 143 CNT2_reg equ Per_base + 060h
FF62 =1 144 MCNT2A_reg equ Per_base + 062h
FF66 =1 145 TMR2CNTR_reg equ Per_base + 066h
=1 146
=1 147 ;Now the memory chip select registers
=1 148
FFA0 =1 149 UMCS_reg equ Per_base + 0A0h ;Upper select
FFA2 =1 150 LMCS_reg equ Per_base + 0A2h ;Lower select
FFA4 =1 151 PACS_reg equ Per_base + 0A4h ;Peripheral select
FFA6 =1 152 MMCS_reg equ Per_base + 0A6h ;Memory select
FFA8 =1 153 MPCC_reg equ Per_base + 0A8h
=1 154
FFCA =1 155 DMA_C0_reg equ Per_base + 0CAh ;DMA control registers
FFDA =1 156 DMA_C1_reg equ Per_base + 0DAh
FFC8 =1 157 DMA_T0_reg equ Per_base + 0C8h ;DMA transfer count registers

```


8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 5

LOC	OBJ	LINE	SOURCE
FFD8	=1	158	DMA_T1_reg equ Per_base + 008h
FFC6	=1	159	DMA_DPH0_reg equ Per_base + 0C6h ;DMA destination high and low
FFD6	=1	160	DMA_DPH1_reg equ Per_base + 0D6h
FFC4	=1	161	DMA_DPL0_reg equ Per_base + 0C4h
FFD4	=1	162	DMA_DPL1_reg equ Per_base + 0D4h
FFC2	=1	163	DMA_SPH0_reg equ Per_base + 0C2h ;DMA source high and low
FFD2	=1	164	DMA_SPH1_reg equ Per_base + 0D2h
FFC0	=1	165	DMA_SPL0_reg equ Per_base + 0C0h
FFD0	=1	166	DMA_SPL1_reg equ Per_base + 0D0h
	=1	167	
	=1	168	; Now the interrupt control registers :
	=1	169	
FF22	=1	170	INT_EOI_reg equ Per_base + 022h ;End of interrupt register
FF24	=1	171	INT_POL_reg equ Per_base + 024h ;Poll register
FF26	=1	172	INT_PST_reg equ Per_base + 026h ;
FF28	=1	173	INT_MSK_reg equ Per_base + 028h ;Mask register
FF2A	=1	174	INT_PRM_reg equ Per_base + 02Ah ;Priority register
FF2C	=1	175	INT_ISR_reg equ Per_base + 02Ch ;In service register
FF2E	=1	176	INT_REQ_reg equ Per_base + 02Eh ;Request register
FF30	=1	177	INT_STS_reg equ Per_base + 030h ;Status
FF32	=1	178	INT_TMC_reg equ Per_base + 032h ;Timer interrupt
FF34	=1	179	INT_DMA0_reg equ Per_base + 034h ;DMA interrupts
FF36	=1	180	INT_DMA1_reg equ Per_base + 036h
FF38	=1	181	INT_INT0_reg equ Per_base + 038h ;External interrupts
FF3A	=1	182	INT_INT1_reg equ Per_base + 03Ah
FF3C	=1	183	INT_INT2_reg equ Per_base + 03Ch
FF3E	=1	184	INT_INT3_reg equ Per_base + 03Eh
	=1	185	
	=1	186	
FE3F	=1	187	UMCS_val equ 0FE3Fh ;Values for the memory registers
3FFB	=1	188	LMCS_val equ 03FFBh ; 4 by 2764s, 512k ram,
007F	=1	189	PACS_val equ 0007Fh ; 3 wait states + ext, peripheral
F9FF	=1	190	MMCS_val equ 0F9FFh ; select block in IO space,
843F	=1	191	MPCS_val equ 0843Fh ; 3 IO wait states + ext.
	=1	192	
8000	=1	193	NS_eoi equ 1000000000000000b ;Non specific end of interrupt
	=1	194	
0005	=1	195	Maxcount0a equ 5 ;Count values for default 9600
0005	=1	196	Maxcount0b equ 5 ; baud serial lines
0005	=1	197	Maxcount1a equ 5
0005	=1	198	Maxcount1b equ 5
C003	=1	199	TMR_baud_control equ 11000000000000011b

25/02/44

```

LOC  OBJ                LINE      SOURCE
                                =1    200      ;
                                =1    201      ; Now the USART equates :
                                =1    202
0400      =1    203      PCS_base      equ      0400h                ;The two USART's on the CPU card
0400      =1    204      USART0_data   equ      PCS_base
0402      =1    205      USART0_control equ      PCS_base + 2
0480      =1    206      USART1_data   equ      PCS_base + 128
0482      =1    207      USART1_control equ      PCS_base + 130
00CA      =1    208      USART_mode_7   equ      11001010b        ; 2 stop bits, no parity, 7 bits
00CE      =1    209      USART_mode_8   equ      11001110b        ; 2 stop bits, no parity, 8 bits
0037      =1    210      USART_cmd      equ      00110111b        ; Enable all!
0040      =1    211      USART_reset    equ      01000000b
000A      =1    212      Dummy_cmd      equ      00001010b
0002      =1    213      Rec_mask       equ      02h              ; Look for RE ready
0001      =1    214      Tr_mask        equ      01h              ; Look for TX ready
0000      =1    215      CR              equ      0Dh
000A      =1    216      LF              equ      0Ah
                                =1    217
                                =1    218      ;Bit defs for the bits in the interrupt mask register
                                =1    219
#          =1    220      Int_bits       record  R1_int:1,T1_int:1,RO_int:1,TO_int:1,
                                =1    221      &          DMA_1_int:1,DMA_0_int:1,Int_bits_dummy:1,TMR_int:1
                                =1    222
                                =1    223      ;Bit defs for the 8251 USART status register
                                =1    224
#          =1    225      USART_bits     record  OSR:1,SYN_det:1,USART_error:3,TX_empty:1,RX_rdy:1,
                                =1    226      &          TX_rdy:1
                                227      ;
                                228      ;
                                229      name      Comms_module
                                230      ;
                                231      ;
                                232      ;Set up a stack segment:
                                233      ;
-----  234      stack      segment      STACK          'STACK'
                                235
0000 (200 236      dw      200      dup      (?)
      ????
      )
                                237
-----  238      stack      ends
                                239      ;
                                240      ;

```


8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 7

LOC	OBJ	LINE	SOURCE
		241	;Now some equates
		242	;
		243	;
E001		244	TMR_clock_cmd equ 1110000000000001b ;Set up a timer as a
		245	;
		246	;
C350		247	Prescale equ 50000 ; interrupts at 30 Hz
		248	;
		249	;
0080		250	F_length equ 128
		251	;
		252	;Now define the characters used in the frames :
		253	;
0010		254	DLE equ 20q
0004		255	End_char equ 4q
0006		256	ACK equ 6q
0015		257	NAK equ 25q
0003		258	CONT equ 3q
0002		259	Data_char equ 2q
		260	;
		261	;Set up the programming of the interrupt control registers
		262	;
0007		263	INT_TMC_val equ 0111b
0018		264	INT_INT0_val equ 011011b
0014		265	INT_INT1_val equ 010100b
		266	;
		267	;Define the event interrupt numbers :
		268	;
0064		269	Frame_rec_int equ 100
0065		270	Time_out_int equ 101
		271	;
		272	;The CRC start and end values :
		273	;
FFFF		274	Crc_init equ 0FFFFh
F0B8		275	Crc_final equ 0F0B8h
		276	;
		277	;
00FF		278	True equ 0FFh
0000		279	False equ 0h
		280	;
		281	; The divisors for the various baud rates :
		282	;
0005		283	Baud_96_a equ 5

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 8

LOC	OBJ	LINE	SOURCE
0005		284	Baud_96_b equ 5
000A		285	Baud_48_a equ 10
000A		286	Baud_48_b equ 10
0014		287	Baud_24_a equ 20
0014		288	Baud_24_b equ 20
0028		289	Baud_12_a equ 40
0028		290	Baud_12_b equ 40
004F		291	Baud_6_a equ 79
004F		292	Baud_6_b equ 79
009E		293	Baud_3_a equ 158
009E		294	Baud_3_b equ 158
0139		295	Baud_15_a equ 313
0139		296	Baud_15_b equ 313
01AC		297	Baud_11_a equ 428
01AC		298	Baud_11_b equ 428
		299	;
		300	;
#		301	Status_flags record Buf_empty:1, Error:1, TX_busy:1, F_ready:1, RX_busy:1
		302	;
		303	
		304	
		305	
		306	;Now for some data!
		307	
----		308	C_data segment 'DATA'
		309	
0000 ????		310	R_f_num dw ? ;Received frame number
0002 ????		311	R_length dw ? ;Received frame length
0004 ????		312	R_in_ptr dw ? ;Next received char pointer
0006 ????		313	R_out_ptr dw ? ;Next processed char
0008 (136 ??)		314	R_data db F_length + 8 dup (?) ;Received frame buffer
0090 ??		315	R_error db ? ;Receive status
0091 ??		316	R_t_char db ? ;Receive frame end char
0092 ??		317	R_f_char db ? ;Rec. frame start char
0093 ??		318	Status db ? ;Status info
0094 ????		319	T_f_num dw ? ;Transmit frame number
0096 ????		320	T_length dw ? ;Tx frame length
0098 (136 ??)		321	T_data db F_length + 8 dup (?) ;Tx frame buffer
0120 (256		322	R_buf db 256 dup (?) ;Rec. char. buffer

8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 9

```

L3C OBJ          LINE    SOURCE
    ??
    )
0220 ??          323      T_t_char      db      ?           ;Tx end character
0221 ??          324      T_f_char      db      ?           ;Tx start char
0222 ????        325      R_state       dw      ?           ;Receive state
0224 ????        326      T_state       dw      ?           ;Tx state variable
0226 ????        327      Tr_index      dw      ?           ;Index into Tx buffer
0228 ????        328      R_crc         dw      ?           ;Received CRC
022A ????        329      T_crc         dw      ?           ;Tx CRC
022C ????        330      T_crc_save    dw      ?           ;Saved CRC value
022E ????        331      Count        dw      ?
    332
    333
----            334      C_data         ends
    335
    336
    337      ; Define a segment where the interrupt vectors are inserted :
----            338      Int_pointers    segment at      0h
    339
004C            340      org      19*4
    341
004C ????        342      Type_19_offset dw      ?
004E ????        343      Type_19_seg    dw      ?
    344
0030            345      org      12*4
    346
0030 ????        347      Type_12_offset dw      ?
0032 ????        348      Type_12_seg    dw      ?
    349
0034 ????        350      Type_13_offset dw      ?
0036 ????        351      Type_13_seg    dw      ?
    352
    353
----            354      Int_pointers    ends
    355
----            356      C_code         segment      'CODE'
    357
    358      assume      CS : C_code
    359
    360      ; Now the state tables (used for indirect jumps)
    361
0000 4004        362      R_table      dw      Rec_dle, Rec_fchar, Rec_fnum, Rec_data, Rec_tchar,
0002 5E04

```


8086/87/88/186 MACRO ASSEMBLER

COMMS_MODULE

04/29/86 PAGE 10

LOC OBJ	LINE	SOURCE
0004 7004		
0006 7604		
0008 9A04		
000A E204	363	& Com_rec_exit, Rec_bcc
000C B604	364	
000E 6105	365	T_table dw Tr_dle, Tr_fchar, Tr_fnum, Tr_data, Tr_data_dle, Tr_dle,
0010 6605		
0012 6C05		
0014 7F05		
0016 A705		
0018 6105		
001A 8105	366	& Tr_tchar, Tr_bcc_lo, Tr_bcc
001C B705		
001E C205	367	
	368	;Now the baud rate selection tables
	369	
0020 AC01	370	Baud_tbl dw Baud_11_a,Baud_11_b,Baud_15_a,Baud_15_b,Baud_3_a,Baud_3_b,
0022 AC01		
0024 3901		
0026 3901		
0028 9E00		
002A 9E00		
002C 4F00	371	& Baud_6_a,Baud_6_b,Baud_12_a,Baud_12_b,Baud_24_a,Baud_24_b,
002E 4F00		
0030 2800		
0032 2800		
0034 1400		
0036 1400		
0038 0A00	372	& Baud_48_a,Baud_48_b,Baud_96_a,Baud_96_b
003A 0A00		
003C 0500		
003E 0500	373	
	374	;Look-up table to get the new CRC
	375	
0040 0000	376	CRC_tbl dw 00000H,01189H,02312H,0329BH,04624H,057A0H,06536H,074BFH
0042 8911		
0044 1223		
0046 9B32		
0048 2446		
004A AD57		

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 11

LOC	OBJ	LINE	SOURCE
004C	3665		
004E	2F74		
0050	488C	377	dw 08C48H,090C1H,0AF5AH,0BED3H,0CA6CH,003E3H,0E37EH,0F8F7H
0052	C19D		
0054	5AAF		
0056	03BE		
0058	6CCA		
005A	E5D8		
005C	7EE9		
005E	F7F8		
0060	8110	378	dw 01081H,00108H,03393H,0221AH,056A5H,0472CH,075B7H,0643EH
0062	0801		
0064	9333		
0066	1A22		
0068	A556		
006A	2C47		
006C	8775		
006E	3E64		
0070	C99C	379	dw 09CC9H,08D40H,0BFDBH,0AE52H,0DAE9H,0CB64H,0F9FFH,0E876H
0072	408D		
0074	08BF		
0076	52AE		
0078	E0DA		
007A	64CB		
007C	FFF9		
007E	76E8		
0080	0221	380	dw 02102H,0308BH,00210H,01399H,06726H,076AFH,04434H,055BDH
0082	8B30		
0084	1002		
0086	9913		
0088	2667		
008A	AF76		
008C	3444		
008E	BD55		
0090	4AAD	381	dw 0AD4AH,0BCC3H,08E58H,09F01H,0EB6EH,0FAE7H,0C87CH,0D9F5H
0092	C3BC		
0094	588E		
0096	D19F		
0098	6EEB		
009A	E7FA		
009C	7CC8		
009E	F5D9		
00A0	8331	382	dw 03183H,0200AH,01291H,00318H,077A7H,0662EH,05485H,0453CH

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 12

LOC	OBJ	LINE	SOURCE
00A2	0A20		
00A4	9112		
00A6	1803		
00A8	A777		
00AA	2E66		
00AC	B554		
00AE	3C45		
00B0	C88D	383	dw 0BDCBH,0AC42H,09ED9H,08F50H,0FBEFH,0EA66H,008F0H,0C974H
00B2	42AC		
00B4	099E		
00B6	508F		
00B8	EFF8		
00BA	56EA		
00BC	F008		
00BE	74C9		
00C0	0442	384	dw 04204H,05380H,06116H,0709FH,00420H,015A9H,02732H,03688H
00C2	8053		
00C4	1661		
00C6	9F70		
00C8	2004		
00CA	A915		
00CC	3227		
00CE	BB36		
00D0	4CCE	385	dw 0CE4CH,0DFC5H,0ED5EH,0FCD7H,08868H,099E1H,0A87AH,0BAF3H
00D2	C5DF		
00D4	5EED		
00D6	07FC		
00D8	6888		
00DA	E199		
00DC	7AAB		
00DE	F3BA		
00E0	8552	386	dw 05285H,0430CH,07197H,0601EH,014A1H,00528H,037B3H,0263AH
00E2	0C43		
00E4	9771		
00E6	1E60		
00E8	A114		
00EA	2805		
00EC	8337		
00EE	3A26		
00F0	CDDE	387	dw 0DEC0H,0CF44H,0F00FH,0EC56H,098E9H,08960H,0BBFBH,0AA72H
00F2	44CF		
00F4	0FF0		
00F6	56EC		

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 13

LOC	OBJ	LINE	SOURCE
00F8	E998		
00FA	6089		
00FC	FBB8		
00FE	72AA		
0100	0663	388	dw 06306H,0728FH,04014H,05190H,02522H,03448H,00630H,01789H
0102	8F72		
0104	1440		
0106	9D51		
0108	2225		
010A	A834		
010C	3006		
010E	B917		
0110	4EEF	389	dw 0EF4EH,0FEC7H,0CC5CH,00005H,0A96AH,088E3H,08A78H,098F1H
0112	C7FE		
0114	5CCC		
0116	05D0		
0118	6AA9		
011A	E3B8		
011C	788A		
011E	F19B		
0120	8773	390	dw 07387H,0620EH,05095H,0411CH,035A3H,0242AH,016B1H,00738H
0122	0E62		
0124	9550		
0126	1C41		
0128	A335		
012A	2A24		
012C	8116		
012E	3807		
0130	CFFF	391	dw 0FFCFH,0EE46H,00CDDH,0CD54H,089EBH,0A862H,09AF9H,08870H
0132	46EE		
0134	000C		
0136	54CD		
0138	EB89		
013A	62A8		
013C	F99A		
013E	708B		
0140	0884	392	dw 08408H,09581H,0A71AH,08693H,0C22CH,003A5H,0E13EH,0F087H
0142	8195		
0144	1AA7		
0146	93B6		
0148	2CC2		
014A	A5D3		
014C	3EE1		

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 14

LOC	OBJ	LINE	SOURCE
014E	B7F0		
0150	4008	393	dw 00840H,019C9H,02B52H,03A0BH,04E64H,05FEDH,06076H,07CFFH
0152	C919		
0154	522B		
0156	0B3A		
0158	644E		
015A	ED5F		
015C	766D		
015E	FF7C		
0160	8994	394	dw 09489H,08500H,0879BH,0A612H,002A0H,0C324H,0F1BFH,0E036H
0162	0085		
0164	98B7		
0166	12A6		
0168	A0D2		
016A	24C3		
016C	8FF1		
016E	36E0		
0170	C118	395	dw 018C1H,00948H,03B03H,02A5AH,05EE5H,04F6CH,07DF7H,06C7EH
0172	4809		
0174	033B		
0176	5A2A		
0178	E55E		
017A	6C4F		
017C	F77D		
017E	7E6C		
0180	0AA5	396	dw 0A50AH,0B483H,08618H,09791H,0E32EH,0F2A7H,0C03CH,0D1B5H
0182	83B4		
0184	1886		
0186	9197		
0188	2EE3		
018A	A7F2		
018C	3CC0		
018E	B5D1		
0190	4229	397	dw 02942H,038CBH,00A50H,01B09H,06F66H,07EEFH,04C74H,050F0H
0192	CB38		
0194	500A		
0196	0918		
0198	666F		
019A	EF7E		
019C	744C		
019E	FD5D		
01A0	88B5	398	dw 0B58BH,0A402H,09699H,0B710H,0F3AFH,0E226H,000B0H,0C134H
01A2	02A4		

8086/87/88/186 MACRO ASSEMBLER COMMS_MODULE

04/29/86 PAGE 15

LOC	OBJ	LINE	SOURCE
01A4	9996		
01A6	1087		
01A8	AFF3		
01AA	26E2		
01AC	B0D0		
01AE	34C1		
01B0	C339	399	dw 039C3H,0284AH,01AD1H,00858H,07FE7H,06E6EH,05CF5H,04D7CH
01B2	4A28		
01B4	011A		
01B6	580B		
01B8	E77F		
01BA	6E6E		
01BC	F55C		
01BE	7C4D		
01C0	0CC6	400	dw 0C60CH,00785H,0E51EH,0F497H,08028H,091A1H,0A33AH,082B3H
01C2	85D7		
01C4	1EE5		
01C6	97F4		
01C8	2880		
01CA	A191		
01CC	3AA3		
01CE	83B2		
01D0	444A	401	dw 04A44H,05BCDH,06956H,078DFH,00C60H,010E9H,02F72H,03EF8H
01D2	CD5B		
01D4	5669		
01D6	DF78		
01D8	600C		
01DA	E91D		
01DC	722F		
01DE	FB3E		
01E0	8DD6	402	dw 0D68DH,0C704H,0F59FH,0E416H,090A9H,03120H,0B3BBH,0A232H
01E2	04C7		
01E4	9FF5		
01E6	16E4		
01E8	A990		
01EA	2081		
01EC	8883		
01EE	32A2		
01F0	C55A	403	dw 05AC5H,04B4CH,079D7H,0685EH,01CE1H,00D68H,03FF3H,02E7AH
01F2	4C4B		
01F4	0779		
01F6	5E68		
01F8	E11C		

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 7
BLOCK_10

STMT LINE NESTING

SOURCE TEXT: :F3:ADMBCN.SRC

```
85  92  0  0      procedure inc(var num : integer);  
                   (* This procedure increments a modulo two frame number. *)  
86  96  1  0      begin  
86  98  1  1          if num = 0 then num := 1  
87  99  1  1          else num := 0;  
89 101  1  1      end;
```

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 8
BLOCK 10

STMT LINE NESTING

SOURCE TEXT: :F3:ADMBCM.SRC

90 107 0 0

procedure Set_next_frame;

(* This procedure set the pointers up for the next data frame. *)

91 111 1 0

begin

91 113 1 1

Out_count := Out_count + Frame_len_out;

92 114 1 1

T_block_len := T_block_len - Max_f_len;

93 115 1 1

if T_block_len < 0 then Frame_len_out := Max_f_len + T_block_len

94 116 1 1

else Frame_len_out := Max_f_len;

96 118 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 9
BLOCK_10

STMT LINE NESTING

SOURCE TEXT: F3:A0MBCM.SRC

```
97 124 0 0      procedure Send_frame;

                (* This procedure sends a data frame. This may be either a new frame as set
                  by Set_next_frame or a retry. *)

98 129 1 0      begin

99 131 1 1          if Frame_diag then write('Send frame', T_fnum);
100 132 1 1          Set_timer(Time_out_val);          (* Set the time out value *)
101 133 1 1          if T_block_len > 0 then T_frame(START, CONT, Frame_len_out, T_fnum, T_buf, Out_count)
102 134 1 1          else T_frame(START, END_ch, Frame_len_out, T_fnum, T_buf, Out_count);

104 136 1 1      end;
```

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 10
BLOCK_ID

STMT LINE NESTING

SOURCE TEXT: :F3:ADMBCH.SRC

105 142 0 0

procedure Init_block_IO;

(* This procedure initializes the module. Note that the lower level module
is also initialized. *)

106 147 1 0

begin

106 149 1 1

Init_coms;

107 150 1 1

T_fnum := 0;

108 151 1 1

R_fnum := 0;

109 152 1 1

R_block_len := 0;

110 153 1 1

SETINTERRUPT(Frame_rec_int, Frame_rec);

111 154 1 1

SETINTERRUPT(Time_out_int, Time_out);

112 156 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 11
BLOCK 10

STMT LINE NESTING

SOURCE TEXT: :F3:ADMBCM.SRC

113 162 0 0

procedure Block_in(var In_block : Input_buf_type);

(* This procedure is called by the main program to transfer a received
block of data. *)

114 167 1 0

begin

114 169 1 1

In_block.Write_block := R_buf;

115 170 1 1

R_block_len := 0;

116 172 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 12
BLOCK 10

STMT LINE NESTING

SOURCE TEXT: :F3:ADMBCM.GRC

```
117 178 0 0      procedure Block_out(var Out_block : Output_buf_type;  
117 179 1 0          Block_length : integer);  
  
      (* This procedure is called by the main program in order to transmit the  
      specified response block. *)  
  
118 184 1 0      begin  
  
118 186 1 1          T_buf := Out_block.Read_block;  
119 187 1 1          T_block_len := Block_length;  
120 188 1 1          Out_count := 0;  
121 189 1 1          Frame_len_out := 0;  
122 190 1 1          Set_next_frame;  
123 191 1 1          Send_frame;  
  
124 193 1 1      end;  
  
      $EJECT
```


SERIES-III Pascal-86, V3.0

06/05/86

PAGE 13
BLOCK 10

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMBCH.SRC

                                $INTERRUPT(Frame_rec)

125 201 0 0             procedure Frame_rec;

                                (* This procedure handles frame received interrupt generated by the frame
                                handler module. *)

126 206 1 0             begin

126 208 1 1               ENABLEINTERRUPTS;          (* We dont want to miss anything. *)
127 209 1 1               R_frame(Start_char, End_char_in, Frame_len_in, R_buf, R_block_len, F_num_in, Error);

128 211 1 1               if Frame_diag then begin

129 213 1 2                 write('Frame rec ');
130 214 1 2                 write(F_num_in);
131 215 1 2                 if Error then write(' error in frame ');
133 216 1 2                 write(' Len ', Frame_len_in);

134 218 1 2               end;

136 220 1 1               case Start_char of

137 222 1 2                 ACK : begin                (* Is it an ACK frame? *)

137 224 1 3                   if Frame_diag then writeln(' ACK frame');
139 225 1 3                   if (not Error) and (End_char_in = END_ch) and (F_num_in = T_fnum) then
140 226 1 3                     begin
140 227 1 4                       Inc(T_fnum);
141 228 1 4                       if T_block_len >= 0 then begin (* Do we want to send another? *)
142 229 1 5                         Set_next_frame;
143 230 1 5                         Send_frame;
144 231 1 5                         end
145 232 1 4                       else Set_timer(0);

147 234 1 4                     end
148 235 1 3                   else Send_frame;

150 237 1 3                 end;

152 239 1 2                 NAK : begin

152 241 1 3                   if Frame_diag then writeln(' NAK frame');
154 242 1 3                   Send_frame;

155 244 1 3                 end;

157 246 1 2                 START : begin

157 248 1 3                   if Frame_diag then writeln(' START frame');
159 249 1 3                   if (Error) then T_frame (NAK, END_ch, 0, F_num_in, T_buf, 0)
160 250 1 3                   else begin

```

SERIES-III Pascal-86, V3.0
FRAME_REC

06/05/86

PAGE 14
BLOCK_10

```
STMT LINE NESTING      SOURCE TEXT: :F3:ADMBCM.SRC
161 252 1 4            T_frame(ACK, END_ch, 0, F_num_in, T_buf, 0);
162 253 1 4            if F_num_in = R_fnum then begin
163 254 1 5              R_block_len := R_block_len + Frame_len_in;
164 255 1 5              Inc(R_fnum);

165 257 1 5              if End_char_in = END_ch then CAUSEINTERRUPT(Block_rec_int);
167 258 1 5              end;
169 259 1 4            end;

171 261 1 3            end;

173 264 1 2            otherwise if Frame_diag then writeln('    Unknown frame type');
176 266 1 2            end;

178 270 1 1            end;
```

\$EJECT

SERIES-III Pascal-86, V3.0

06/05/86

PAGE 15

BLOCK_ID

STMT LINE NESTING SOURCE TEXT: :F3:ADMBCM.SRC

\$INTERRUPT(Time_out)

179 279 0 0

procedure Time_out;

(* This procedure handles timeouts. The previous frame is simply retransmitted. *)

180 283 1 0

begin

180 285 1 1

if Frame_diag then writeln(' Time out');

182 286 1 1

Send_frame;

183 288 1 1

end;

Summary Information:

PROCEDURE	OFFSET	CODE SIZE	DATA SIZE	STACK SIZE
INIT_BLOCK_ID	015AH	003FH	630	000EH 140
BLOCK_IN	0199H	0010H	290	0006H 60
BLOCK_OUT	0186H	0035H	530	000AH 100
INC	0078H	0025H	370	0006H 60
SET_NEXT_FRAME	0090H	0033H	510	0006H 60
SEND_FRAME	00D0H	008AH	1380	0016H 220
FRAME_REC	01EBH	0268H	6190	0032H 500
TIME_OUT	0456H	0047H	710	0026H 380
-CONST IN CODE-		0078H	1200	
Total		0490H 11810	28C5H 104370	0098H 1520

427 Lines Read.

0 Errors Detected.

Dictionary Summary:

120KB Memory Available.

10KB Memory Used (8%).

0KB Disk Space Used.

6KB out of 16KB Static Space Used (37%).

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 1

Source File: :F3:ADMTN.SRC
 Object File: :F3:ADMTN.OBJ
 Controls Specified: DEBUG, OPTIMIZE(0).

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTN.SRC

```
(*****  

  *****)
```

MODULE FUNCTION :

This module is the main program for the stand alone tape controller.

```
*****
```

Written : A.D.McGuffog

22 October 1985.

```
*****
```

Language : Intel Pascal-86 V3.0.

```
*****
```

Other Software Required : Comms_module, Tape_controller_primitives.

```
*****
```

Restrictions : None.

```
*****
```

Module description :

This module provides the following functions :

1. Allows the operating parameters (baud rates, tape transport speeds, terminal types etc) of the controller to be set.
2. Allow the user to operate the tape transports in local mode. Tapes may be read, written, erased or copied without host system intervention.
3. Provides an interface to the PDP-11/23 host system. Block, each block corresponding to a single PDP-11 IO request, are transmitted to the tape controller. These are then interpreted by this program and a reply block returned. Received blocks are signaled by a software interrupt.

```
*****)
```

```
$EJECT
```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 2

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```

$large(Fixup has Tape_controller_primitives;
$      exports Init_tape, Select, Erase_tape, Read_block;
$      exports Write_block, Write_with_long_gap, Write_EOF;
$      exports Space, Cntrl_status, Off_line, Rewind_tape;
$      exports Set_RTH_high, Set_RTH_low, Diag;
$      has Comms_module;
$      exports Init_coms, Set_baud, R_frame, T_frame;
$      exports Set_timer)

```

```

$compact(-const in code-)

```

```

1  58 0 0 module Tape_controller_main;

    =1
2  2 0 0 =1 $include(:f3:admpdf.src)
    =1
3  4 0 0 =1 public Tape_controller_primitives;
    =1
4  6 0 0 =1     procedure Init_tape;
    =1
5  9 0 0 =1     procedure Select(Unit_sel : integer;
    =1               T_data : Tape_data_type);
    =1
6  11 0 0 =1     procedure Erase_tape;
    =1
7  14 0 0 =1     procedure Read_block(var Read_buf : buffer;
    =1               var N_r_char : integer);
    =1
8  17 0 0 =1     procedure Write_block(var Write_buf : buffer;
    =1               N_w_char : integer);
    =1
9  20 0 0 =1     procedure Write_with_long_gap(var Write_buf : buffer;
    =1               N_w_char : integer);
    =1
10 22 0 0 =1     procedure Write_EOF;
    =1
11 24 0 0 =1     procedure Space(var Sp_count : integer);
    =1
12 26 0 0 =1     procedure Cntrl_status(var Status_return : Status_type);
    =1
13 28 0 0 =1     procedure Off_line;
    =1
14 30 0 0 =1     procedure Rewind_tape;
    =1
15 32 0 0 =1     procedure Set_RTH_high;
    =1
16 34 0 0 =1     procedure Set_RTH_low;
    =1
17 39 0 0 =1     procedure Diag(var R_buf : buffer;
    =1               Index : integer;
    =1               var Parity_error : boolean;
    =1               var State : integer);
    =1
17 39 0 0 =1 public Comms_module;

```

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 3
TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMPDF.SRC
      =1
18  41  0  0 =1      procedure Init_coms;
      =1
19  43  0  0 =1      procedure Set_baud(Chan_number : integer;
      =1                      Rate_select : integer);
      =1
20  46  0  0 =1      procedure R_frame(var T_char : integer;
      =1                      var F_char : integer;
      =1                      var F_length : integer;
      =1                      var F_data : Block;
      =1                      F_index : integer;
      =1                      var F_num : integer;
      =1                      var F_error : boolean);
      =1
21  54  0  0 =1      procedure T_frame(T_f_char : integer;
      =1                      T_t_char : integer;
      =1                      T_length : integer;
      =1                      T_f_num : integer;
      =1                      var T_data : Block;
      =1                      T_index : integer);
      =1
22  61  0  0 =1      procedure Set_timer(Time : integer);
      =1
23  64  0  0 =1      public Block_ID;
      =1
24  66  0  0 =1          const  Frame_rec_int      = 100;
25  67  0  0 =1          Time_out_int      = 101;
26  68  0  0 =1          Block_rec_int      = 102;
      =1
27  71  0  0 =1      procedure Init_block_ID;
      =1
28  73  0  0 =1      procedure Block_in(var In_block : Input_buf_type);
      =1
29  75  0  0 =1      procedure Block_out(var Out_block : Output_buf_type;
      =1                      Block_length : integer);
      =1
30  80  0  0 =1      public Tape_controller_main;
      =1
31  82  0  0 =1          const  Data_length      = 5201;
32  83  0  0 =1          Block_length      = 5207;
33  84  0  0 =1          N_drives      = 1;
34  85  0  0 =1          Clk_den      = 6250;
35  86  0  0 =1          Unit_0_MC      = 139;
36  87  0  0 =1          Unit_1_MC      = 250;
      =1
37  90  0  0 =1      type  Tape_data_type = record
      =1
37  92  0  1 =1                      Stop : integer;
38  93  0  1 =1                      Master_clock : integer;
39  94  0  1 =1                      Long_gap : integer;

```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 4

TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMPDF.SRC
 40  95  0  1 =1      Short_gap : integer;
 41  96  0  1 =1      Start_norm : integer;
 42  97  0  1 =1      Start_long : integer;
 43  98  0  1 =1      Start_EOF : integer;
 44  99  0  1 =1      Start_read : integer;
                        =1
 45 101  0  1 =1      end;
                        =1
                        =1
 46 104  0  0 =1      Buffer      = packed array [0..Data_length] of char;
                        =1
 47 106  0  0 =1      Block      = packed array [0..Block_length] of char;
                        =1
 48 108  0  0 =1      Status_type = set of (P_err, CRC_err, Length_err, DMA_err,
                        =1      EDT, EDF, SWL, IGX, S_err, RWD, HWL,
                        =1      xx, Time_err, BOT, ECV, PEDV);
                        =1
 49 112  0  0 =1      Input_buf_type = record case boolean of
                        =1
 50 114  0  1 =1          true      : (Opcode      : char;
 51 115  0  1 =1          Unit_num   : char;
 52 116  0  1 =1          Tape_status_in : Status_type;
 53 117  0  1 =1          Count_in    : integer;
 54 118  0  1 =1          Write_data   : Buffer);
                        =1
 55 120  0  1 =1          false     : (Write_block : Block);
                        =1
 56 122  0  1 =1      end;
                        =1
                        =1
 57 125  0  0 =1      Output_buf_type = record case boolean of
                        =1
 58 127  0  1 =1          true      : (Tape_status_out : Status_type;
 59 128  0  1 =1          Count_out  : integer;
 60 129  0  1 =1          Ret_code   : integer;
 61 130  0  1 =1          Read_data   : Buffer);
                        =1
 62 132  0  1 =1          false     : (Read_block    : Block);
                        =1
 63 134  0  1 =1      end;
                        =1
 64 136  0  0 =1      var      Block_diag : boolean;
 65 137  0  0 =1      Frame_diag : boolean;
                        =1
                        =1

```

#EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 5

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```

66 65 0 0      program   Tape_controller_main(input, output);
67 66 0 0      const    IE_DNR      = 375q;      (* These are PDP-11 status return codes. *)
68 67 0 0          IE_FHE      = 305q;
69 68 0 0          IE_BBE      = 310q;
70 69 0 0          IE_DAO      = 363q;
71 70 0 0          IE_EOF      = 366q;
72 71 0 0          IE_EOT      = 302q;
73 72 0 0          IE_EDV      = 365q;
74 73 0 0          IE_SPC      = 372q;
75 74 0 0          IE_VER      = 374q;
76 75 0 0          IE_WLK      = 364q;
77 76 0 0          IE_IFC      = 376q;
78 77 0 0          IS_SUC      = 1;

(* Now the various possible PDP 10 request command codes. *)

79 81 0 0          RLB_cmd      = 1;
80 82 0 0          WLB_cmd      = 2;
81 83 0 0          EOF_cmd      = 3;
82 84 0 0          RWD_cmd      = 4;
83 85 0 0          RWU_cmd      = 5;
84 86 0 0          SPB_cmd      = 6;
85 87 0 0          SPF_cmd      = 7;
86 88 0 0          STC_cmd      = 8;
87 89 0 0          SEC_cmd      = 9;
88 90 0 0          SMO_cmd      = 10;
89 91 0 0          RRB_cmd      = 11;
90 92 0 0          ERS_cmd      = 12;

91 94 0 0          Max_retrys   = 10;      (* Number of times to retry a tape operation. *)

92 96 0 0          blanks      = '          ';

93 98 0 0          type    Out_string = packed array [1..32] of char;

94 100 0 0         var    Unit      : integer;
95 101 0 0          Tape_data  : array [0..N_drives] of Tape_data_type;
96 102 0 0          Unit_table : array [0..N_drives] of integer;
97 103 0 0          Input_buf  : Input_buf_type;
98 104 0 0          Output_buf  : Output_buf_type;
99 105 0 0          Tape_status : Status_type;
100 106 0 0         LUN        : integer;
101 107 0 0         Error_bits  : Status_type;
102 108 0 0         Change_bits : Status_type;
103 109 0 0         Code_errors : Status_type;
104 110 0 0         Hold_bits   : Status_type;
105 111 0 0         Status_bits : Status_type;
106 112 0 0         ANSI        : boolean;      (* Is the tape ANSI format ? *)
107 113 0 0         Last_op_EOF : boolean;
108 114 0 0         Retry_cnt   : integer;
109 115 0 0         Data_size   : integer;
110 116 0 0         Term_type   : integer;

```

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 6
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTN.SRC

#EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 7

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADNTON.SRC

```
111 124 0 0      procedure Display_status(Out_status : Status_type);

                  (* This procedure displays the status of the currently selected tape transport. *)

112 128 1 0      begin

112 130 1 1          write('Unit ', LUN, ' status :');
113 131 1 1          if S_err in Out_status then write(' Select error')
114 132 1 1              else write(' On line');
116 133 1 1          if BOT in Out_status then write(', at BOT')
117 134 1 1              else if EOT in Out_status then write(' past EOT');
120 135 1 1          if EOF in Out_status then write(', EOF detected');

122 137 1 1          if RWD in Out_status then write(', Rewinding');

124 139 1 1          if HWL in Out_status then write(', write locked');
126 140 1 1          if SWL in Out_status then write(', software write lock');
128 141 1 1          if Time_err in Out_status then write(', Time error');
130 142 1 1          if P_err in Out_status then write(', parity error');
132 143 1 1          if CRC_err in Out_status then write(', CRC error');
134 144 1 1          if Length_err in Out_status then write(', length error');
136 145 1 1          if DMA_err in Out_status then write(', DMA error');
138 146 1 1          writeln;

139 148 1 1      end;

                  $EJECT
```

SERIES-III Pascal-36, V3.0

16/06/86

PAGE 6

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

140 153 0 0

function Cap(c:char):char;

(* This function returns the the upper case form of any letter. *)

141 157 1 0

begin.

141 158 1 1

if c > 'Z' then Cap := chr(ord(c) - (ord('a') - ord('A')))

142 159 1 1

else Cap := c;

144 160 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 9

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

145 166 0 0

function Val(c:char):integer;

(* This function return the integer value of any character in the hex. set. *)

146 170 1 0

begin

146 171 1 1

c := Cap(c);

147 172 1 1

if c < 'A' then Val := ord(c) - ord('0')

148 173 1 1

else Val := ord(c) - ord('A') + 10;

150 174 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 10

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

```

151 180 0 0      function   In_int(var num:integer):boolean;

(* This function reads in an integer in one of five formats :

    1. Binary (number ends in 'b').
    2. Octal (number ends in 'o' or 'q').
    3. Decimal (number ends in a digit).
    4. Hex (number ends in 'h').
    5. In the form of a character enclosed in single quotes. Note that in
       this case the ASCII value of the character is returned.

If a carriage return is entered then the number will stay at its original
value. The number is checked for valid format, and if it is valid the the
function returns true. Note that an automatic upper case conversion is
performed (except for character literals). *)

152 200 1 0      var logical_blank   : set of ' '.. '0';
153 201 1 0      terminator         : set of ' '.. 'Q';
154 202 1 0      inchar             : array [1..32] of char;
155 203 1 0      temp               : longint;
156 204 1 0      i,j,base           : integer;

157 207 1 0      begin
157 208 1 1        logical_blank := [' ', '0', '1'];
158 209 1 1        terminator := [' ', '1', 'H', 'Q', 'C', 'B'];
159 210 1 1        In_int := true;
160 211 1 1        i := 1;
161 212 1 1        inchar[1] := ' ';

162 214 1 1        while not eoln and (inchar[i] in logical_blank) do read(inchar[i]);
164 215 1 1        if not (eoln and (inchar[i] = ' ')) then

165 217 1 1          if inchar[i] = ''' then begin
166 218 1 2            if not eoln then read(inchar[2]);
168 219 1 2            if not eoln then begin
169 220 1 3              read(inchar[3]);
170 221 1 3              if inchar[3] = ''' then num := ord(inchar[2])
171 222 1 3              else In_int := false;
173 223 1 3              end(*if*);
174 224 1 2            else In_int := false;
176 225 1 2            end(*if*);
177 226 1 1          else begin
178 227 1 2            while not eoln and not (Cap(inchar[i]) in terminator) and (i < 32) do begin
179 228 1 3              i := i + 1;
180 229 1 3              read(inchar[i]);
181 230 1 3              end(*while*);
183 231 1 2            if i > 31 then In_int := false
184 232 1 2            else begin

```

SERIES-III Pascal-86, V3.0
IN_INT

16/06/86 PAGE 11
TAPE_CONTROLLER_MAIN

STMT	LINE	NESTING	SOURCE TEXT: :F3:ADMTN.SRC
185	233	1 3	case Cap(inchar[i]) of
186	234	1 4	'H' : base := 16;
187	235	1 4	'O' : base := 8;
188	236	1 4	'Q' : base := 8;
189	237	1 4	'B' : base := 2;
190	238	1 4	otherwise base := 10;
192	239	1 4	end (*case*);
194	240	1 3	temp := 0;
195	241	1 3	if (base <> 10) and (i <> 1) then i := i - 1;
197	242	1 3	for j := 1 to i do begin
198	243	1 4	i := Val(inchar[j]);
199	244	1 4	if (i < base) and (i >= 0) and (temp < MAXINT) then
200	245	1 4	temp := base + temp + i
			else In_int := false;
202	247	1 4	end(*for*);
204	248	1 3	end (*else*);
206	249	1 2	if temp < MAXINT then num := temp
207	250	1 2	else In_int := false;
209	251	1 2	end;
211	252	1 1	readin;
212	253	1 1	end(*proc*);

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 12

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: F3:ADNTCH.SRC

213 259 0 0

function Get_int_in_range(var num : integer; low, high : integer) : boolean;

(* This function reads in a number as described for In_int, but also checks that
it is within a specified range. *)

214 264 1 0

begin

214 266 1 1

Get_int_in_range := In_int(num);

215 267 1 1

if (num < low) or (num > high) then Get_int_in_range := false;

217 269 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 13

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```
218 276 0 0      function  Get_char_in_range(var num_char : char; low, high : integer) : boolean;

                  (* This function reads in a character as described for In_int, but also checks
                  that it is within a specified range. *)

219 281 1 0      var      num      : integer;
220 282 1 0      begin

220 284 1 1          num := ord(num_char);
221 285 1 1          Get_char_in_range := true;
222 286 1 1          if Get_int_in_range(num, low, high) then num_char := chr(num)
223 287 1 1          else Get_char_in_range := false;

225 289 1 1      end;

$EJECT
```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 14

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```
226 294 0 0      procedure Get_value(prompt : Out_string;
226 295 1 0          var in_var : integer;
                    get_low, get_high : integer);

    (* This procedure reads in a number as described for In_int, but also checks that
    it is within a specified range. A prompt string can also be specified, and the
    procedure does not return until a valid number has been entered. *)

227 302 1 0      var    get_retry : boolean;

228 304 1 0      begin

228 306 1 1          get_retry := false;
229 307 1 1          repeat

229 309 1 2              if get_retry then writeln(' Input out of range. ');
231 310 1 2              write(prompt, ' (', get_low, '..', get_high, ') <', in_var, ') : ');
232 311 1 2              get_retry := true;
233 312 1 2              until Get_int_in_range(in_var, get_low, get_high);

235 314 1 1      end;

$EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 15

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

236 319 0 0

procedure Clr_screen;

(* This procedure clears the terminal screen by means of control codes. Four terminal types are catered for, including the case of an unknown terminal, in which case no action is taken. *)

237 325 1 0

const esc = 27;

238 327 1 0

var i : integer;

239 329 1 0

begin

239 331 1 1

case Term_type of

240 333 1 2

1 : begin

240 335 1 3

write(chr(esc), 'K'); (*Perkin*)

241 336 1 3

for i := 1 to 20 do write(chr(0));

243 338 1 3

end;

245 340 1 2

2 : write(chr(esc), 'CH', chr(esc), 'CJ'); (*ANSI*)

246 342 1 2

3 : write(chr(esc), 'H', chr(esc), 'J'); (*VT52*)

247 344 1 2

otherwise writeln;

249 347 1 2

end;

251 350 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 16

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: IF3:ADMTCN.SRC

252 356 0 0
252 357 1 0

```
function Menu(procedure out_proc(var max : integer);
    disp_flag : boolean;
    disp_string : Out_string;
    disp_int : integer) : integer;
```

(* This function obtain the users choice from a menu. The menu text is displayed by the procedure parameter. If disp_flag is true then disp_string is displayed on the same line as the prompt. If an invalid choice is made then the menu is redisplayed, however the string is replaced by one informing the user that the previous input was in error. Note that the called procedure must return a value denoting the highest possible valid menu choice. *)

253 368 1 0
254 369 1 0

```
var    h, selection : integer;
    repeat_msg      : boolean;
```

255 371 1 0

begin

255 373 1 1

repeat_msg := false;

256 374 1 1

repeat

256 375 1 2

clr_screen;

257 376 1 2

if repeat_msg then begin

258 377 1 3

disp_string := ' Input out of range...try again ';

259 378 1 3

disp_flag := false;

260 379 1 3

end;

262 380 1 2

out_proc(h);

263 381 1 2

write(disp_string);

264 382 1 2

if disp_flag then writeln(disp_int)

265 383 1 2

else writeln;

267 384 1 2

repeat_msg := true;

268 385 1 2

write(' Input your selection (1..', h, '):');

269 386 1 2

selection := 0;

270 387 1 2

until Get_int_in_range(selection, 1, h);

272 388 1 1

menu := selection;

273 390 1 1

end;

#EJECT

SERIES-III Pascal-66, V3.0

16/06/86

PAGE 17

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

```
274 396 0 0      procedure Main_menu(var max_sel : integer);

      (* This is the local mode menu display procedure, and is used in as a parameter
      for the Menu function. *)

275 401 1 0      begin

275 403 1 1          writeln;
276 404 1 1          writeln;
277 405 1 1          writeln('                LOCAL MODE MENU');
278 406 1 1          writeln;
279 407 1 1          writeln;
280 408 1 1          writeln('      1. Copy an unformatted tape.');
```

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 18

TAPE_CONTROLLER.MAIN

STMT LINE NESTING

SOURCE TEXT: F3:ADMTON.SRC

```

295 429 0 0      procedure System_menu(var max_sel : integer);

(* This is the system function menu display procedure, and is used in as a parameter
for the Menu function. *)

296 434 1 0      begin

296 436 1 1          writeln;
297 437 1 1          writeln;
298 438 1 1          writeln('                SYSTEM FUNCTIONS MENU');
299 439 1 1          writeln;
300 440 1 1          writeln;
301 441 1 1          writeln('      1. Set data link baud rate.');
```

- 302 442 1 1 writeln;
- 303 443 1 1 writeln(' 2. Set terminal baud rate.');
- 304 444 1 1 writeln;
- 305 445 1 1 writeln(' 3. Set terminal type.');
- 306 446 1 1 writeln;
- 307 447 1 1 writeln(' 4. Set tape unit characteristics.');
- 308 448 1 1 writeln;
- 309 449 1 1 writeln(' 5. Enable block diagnostics.');
- 310 450 1 1 writeln;
- 311 451 1 1 writeln(' 6. Disable block diagnostics.');
- 312 452 1 1 writeln;
- 313 453 1 1 writeln(' 7. Enable frame diagnostics.');
- 314 454 1 1 writeln;
- 315 455 1 1 writeln(' 8. Disable frame diagnostics.');
- 316 456 1 1 writeln;
- 317 457 1 1 writeln(' 9. Return to local mode menu.');
- 318 458 1 1 writeln;
- 319 459 1 1 writeln;
- 320 460 1 1 max_sel := 9;

```

321 462 1 1      end;

$EJECT

```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 19

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCH.SRC

```
322 468 0 0      procedure Term_menu(var max_sel : integer);

                  (* This is the terminal type menu display procedure, and is used in as a parameter
                  for the Menu function. *)

323 473 1 0      begin

323 475 1 1          writeln;
324 476 1 1          writeln;
325 477 1 1          writeln('                Terminals available are :');
326 478 1 1          writeln;
327 479 1 1          writeln;
328 480 1 1          writeln('      1. Perkin Elmer.');
```

```
329 481 1 1          writeln;
330 482 1 1          writeln('      2. VT100/ANSI.');
```

```
331 483 1 1          writeln;
332 484 1 1          writeln('      3. VT52.');
```

```
333 485 1 1          writeln;
334 486 1 1          writeln('      4. Other.');
```

```
335 487 1 1          writeln;
336 488 1 1          max_sel := 4;

337 490 1 1      end;

$EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 20

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```
338 496 0 0      procedure Baud_menu(var max_sel : integer);

                (* This is the baud rate menu display procedure, and is used in as a parameter
                for the Menu function. *)

339 501 1 0      var      i      : integer;

340 503 1 0      begin

340 505 1 1          writeln;
341 506 1 1          writeln;
342 507 1 1          writeln('                Baud rates available are :');
343 508 1 1          writeln;
344 509 1 1          writeln;
345 510 1 1          writeln('      1. 110 baud. ');
346 511 1 1          writeln;
347 512 1 1          writeln('      2. 150 baud. ');
348 513 1 1          writeln;
349 514 1 1          writeln('      3. 300 baud. ');
350 515 1 1          writeln;
351 516 1 1          writeln('      4. 600 baud. ');
352 517 1 1          writeln;
353 518 1 1          writeln('      5. 1200 baud. ');
354 519 1 1          writeln;
355 520 1 1          writeln('      6. 2400 baud. ');
356 521 1 1          writeln;
357 522 1 1          writeln('      7. 4800 baud. ');
358 523 1 1          writeln;
359 524 1 1          writeln('      8. 9600 baud. ');
360 525 1 1          writeln;
361 526 1 1          writeln;
362 527 1 1          writeln;
363 528 1 1          max_sel := 8;

364 530 1 1      end;

$EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 21
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```

365 537 0 0      procedure Local_ops_menu(var max_sel : integer);

(* This is the local operations menu display procedure, and is used in as a
parameter for the Menu function. Note that this also displays the tape controller
status. *)

366 543 1 0      begin

366 545 1 1          writeln;
367 546 1 1          writeln('                LOCAL OPERATIONS MENU');
368 547 1 1          writeln;
369 548 1 1          writeln;
370 549 1 1          writeln(' 1. Select a unit. ');
371 550 1 1          writeln(' 2. Read a block into the local buffer. ');
372 551 1 1          writeln(' 3. Write the local buffer to tape. ');
373 552 1 1          writeln(' 4. Write to tape with a long inter-record gap. ');
374 553 1 1          writeln(' 5. Write an EOF mark. ');
375 554 1 1          writeln(' 6. Space blocks forward. ');
376 555 1 1          writeln(' 7. Space blocks reverse. ');
377 556 1 1          writeln(' 8. Set the read threshold high. ');
378 557 1 1          writeln(' 9. Set the read threshold low. ');
379 558 1 1          writeln('10. Rewind. ');
380 559 1 1          writeln('11. Set unit off-line. ');
381 560 1 1          writeln('12. Display the local buffer. ');
382 561 1 1          writeln('13. Modify the local buffer. ');
383 562 1 1          writeln('14. Display local buffer diagnostics information. ');
384 563 1 1          writeln('15. Change the present buffer size. ');
385 564 1 1          writeln('16. Exit to the main menu. ');
386 565 1 1          writeln;

387 567 1 1          Cntrl_status(Tape_status);

388 569 1 1          Display_status(Tape_status);

389 571 1 1          max_sel := 16;

390 573 1 1      end;

```

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 22

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTN.SRC

```
391 579 0 0      procedure Hex_digit_out(hex_digit : integer);  
                  (* This procedure writes a single hex digit out. *)  
392 582 1 0      begin  
392 584 1 1          if hex_digit > 9 then write(chr(hex_digit - 10 + ord('A')))  
393 585 1 1          else write(hex_digit);  
395 587 1 1      end;
```

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 23
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCH.SRC

```
396 594 0 0      procedure Disp_hex(data_byte : char);  
  
                  (* This procedure displays a character value as a hex number. *)  
  
397 598 1 0      begin  
  
397 600 1 1          Hex_digit_out((ord(data_byte) div 16));  
398 601 1 1          Hex_digit_out((ord(data_byte) mod 16));  
399 602 1 1          write(' ');  
  
400 604 1 1      end;
```

*EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 24

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADNTCN.SRC

```
401 610 0 0      function Ascii_char(out_char : char) : char;

                  (* This function returns characters in a form suitable for display on a
                  terminal screen. All non printable characters are displayed as '.'
                  characters. *)

402 616 1 0      begin

403 618 1 1          if (out_char > ' ') and (out_char <= '9') then Ascii_char := out_char
403 619 1 1          else Ascii_char := '.';

405 621 1 1      end;

$EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 25

TAPE_CONTROLLER_MAIN

```
STMT LINE NESTING      SOURCE TEXT: :F3:40MTCN.SRC

406 626 0 0      procedure Set_unit_num;

                    (* This procedure prompts the user for a unit number, and then sets it. *)

407 630 1 0      var Cmd_retry : boolean;

408 632 1 0      begin
408 633 1 1          Cmd_retry := false;
409 634 1 1          clr_screen;
410 635 1 1          repeat

410 637 1 2              if Cmd_retry then writeln(' Unit number out of range. ');
412 638 1 2              write('   Input unit number (0..', N_drives, '): ');
413 639 1 2              Cmd_retry := true;
414 640 1 2          until Get_int_in_range(LUN, 0, N_drives);
416 641 1 1          Unit := Unit_table[LUN];
417 642 1 1          Select(Unit, Tape_data[Unit]);

418 644 1 1      end;

$EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 26

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

419 649 0 0

procedure Bs_read;

(* This procedure executes a single block backspace (unless the unit is at BOT),
then a read operation. *)

420 654 1 0

var i : integer;

421 656 1 0

begin

421 658 1 1

Data_size := Input_buf.Count_in;

422 659 1 1

i := -1;

423 660 1 1

Ctrl_status(Tape_status);

424 661 1 1

if not (BOT in Tape_status) then Space(i);

426 662 1 1

Read_block(Output_buf.Read_data, Data_size);

427 663 1 1

Ctrl_status(Tape_status);

428 665 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 27

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

```

429 670 0 0      procedure Write_ver;

(* This procedure executes a verified write operation. The tape is written then
checked and, if necessary, retried until the block is correctly written or
the maximum number of retrys is reached. If the maximum number of retrys are
reached then a write with a long (3 inch) gap is attempted unless this is not
allowed (IGX set). *)

430 678 1 0      var i          : integer;
431 679 1 0      End_wr_ver    : boolean;

432 681 1 0      begin

432 683 1 1          repeat

432 685 1 2              Data_size := Input_buf.Count_in;
433 686 1 2              Retry_cnt := Retry_cnt - 1;
434 687 1 2              if Last_op_EOF then Write_with_long_gap(Input_buf.Write_data, Data_size)
435 688 1 2              else Write_block(Input_buf.Write_data, Data_size);
437 689 1 2              Bs_read;
438 690 1 2              if (Tape_status*Error_bits <> []) and (Retry_cnt <> 0) then begin
439 691 1 3                  End_wr_ver := false;
440 692 1 3                  i := -1;
441 693 1 3                  Space(i);
442 694 1 3                  end
443 695 1 2              else End_wr_ver := true;

445 697 1 2          until End_wr_ver;

447 699 1 1          if (Tape_status*Error_bits <> []) and (not (IGX in Input_buf.Tape_status_in)) then begin

448 701 1 2              i := -1;
449 702 1 2              Space(i);
450 703 1 2              Write_with_long_gap(Input_buf.Write_data, Data_size);
451 704 1 2              Bs_read;

452 706 1 2          end;

454 708 1 1      end;

$EJECT

```

SERIES-III Pascal-86, V3.0

10/06/86 PAGE 28
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

455 714 0 0

procedure Read_var;

(* This procedure executes a read operation with retry. Reads are attempted until
the block is correctly read or the maximum number of retries is exceeded. *)

456 719 1 0

begin

456 721 1 1

Data_size := Input_buf.Count_in;

457 722 1 1

Read_block(Output_buf.Read_data, Data_size);

458 723 1 1

Cntrl_status(Tape_status);

459 725 1 1

while (Tape_status*Error_bits <> 0) and (Retry_cnt <> 0) do begin

460 727 1 2

Bs_read;

461 728 1 2

Retry_cnt := Retry_cnt - 1;

462 730 1 2

end;

464 732 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 29
TAPE_CONTROLLER_MAIN

```
STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC

465 737 0 0            procedure Set_code;

                        (* This procedure check for EOF and EDT, and sets the return status as
                           appropriate. *)

466 742 1 0            begin

466 744 1 1              if (EDT in Tape_status) then Output_buf.Ret_code := IE_EDT;
468 745 1 1              if (EOF in Tape_status) then Output_buf.Ret_code := IE_EOF;

470 747 1 1            end;

                        $EJECT
```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 30

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

471 753 0 0

procedure Ver_check;

(* This procedure checks for data errors and sets the return code as appropriate. *)

472 758 1 0

begin

472 760 1 1

if (Tape_status*Code_errors <> []) then Output_buf.Ret_code := IE_VER;

474 761 1 1

if (Length_err in Tape_status) then Output_buf.Ret_code := IE_DAO;

476 763 1 1

end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 31
TAPE_CONTROLLER_MAIN

```
STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC

477 769 0 0            procedure EDV_test;

                        (* This procedure checks for the logical end of volume condition. This is done
                           for unlabeled tapes only as for ANSI format tapes EDV is denoted by EDV
                           records, not a double tape mark. If the tape is at EDV then it is positioned
                           between the tape marks. *)

478 776 1 0            var i          : integer;

479 778 1 0            begin

479 780 1 1              if (not ANSI) and (Last_op_EOF) and (EOF in Tape_status) then begin

480 782 1 2                Tape_status := Tape_status + [EDV];
481 783 1 2                Output_buf.Ret_code := IE_EDV;
482 784 1 2                i := -1;
483 785 1 2                Space(i);
484 786 1 2                Last_op_EOF := true;

485 788 1 2              end;

487 790 1 1            end;

                        $EJECT
```

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 32

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```

488 795 0 0      procedure Wipe_tape;

(* This procedure erases a tape on one of the transports from the BOT tab to
the EDT tab. The user is prompted for the unit number, and then prompted to
install the tape on the unit. The unit is checked for interlocks, and the user
prompted if necessary. The unit is then rewound, erased and then rewound
again. *)

489 803 1 0      var      erase_end      : boolean;
490 804 1 0      unit_ready : boolean;

491 806 1 0      begin

491 808 1 1          Set_unit_num;
492 809 1 1          erase_end := false;

493 811 1 1          repeat
493 812 1 2              clr_screen;
494 813 1 2              writeln;
495 814 1 2              writeln;
496 815 1 2              writeln('      Mount tape on selected unit.');
```

497 816 1 2 writeln;

498 817 1 2 writeln(' Hit return when ready, c then return to cancel.');

499 818 1 2 writeln;

500 819 1 2 if not eofn then erase_end := true;

502 820 1 2 readln;

503 822 1 2 if not erase_end then begin

504 824 1 3 Cntrl_status(Tape_status);

505 825 1 3 unit_ready := true;

506 826 1 3 if S_err in Tape_status then begin

507 828 1 4 writeln(' The selected unit is not ready.');

508 829 1 4 writeln;

509 830 1 4 unit_ready := false;

510 832 1 4 end(*if*);

512 834 1 3 if HWL in Tape_status then begin

513 836 1 4 writeln(' The selected unit is not write enabled.');

514 837 1 4 writeln;

515 838 1 4 unit_ready := false;

516 840 1 4 end(*if*);

518 841 1 3 Rewind_tape;

519 843 1 3 if unit_ready then begin

520 845 1 4 writeln;

521 846 1 4 writeln(' Working. . . .');

522 847 1 4 Erase_tape;

SERIES-III Pascal-86, V3.0
WIPE_TAPE

16/06/86 PAGE 33
TAPE_CONTROLLER_MAIN

```
STMT LINE NESTING      SOURCE TEXT: :F3:ADMTN.SRC
523 848 1 4            Rewind_tape;
524 849 1 4            erase_end := true;

525 851 1 4            end;

527 853 1 3            end;

529 855 1 2            until erase_end;

531 857 1 1            end;

                        *EJECT
```

SERIES-III Pascal-66, V3.0

16/06/86

PAGE 34

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: IF3:ADMTCN.SRC

```

532 863 0 0      procedure Tape_copy;

(* This procedure copies a tape from logical unit 0 to logical unit 1. The user
is prompted to install the source and destination tapes. The units are then
checked for interlocks, and the user prompted if necessary. The units are then
rewound, copied and rewound again. *)

533 870 1 0      const  source_unit = 1;
534 871 1 0      dest_unit  = 0;

535 874 1 0      var    count      : integer;
536 875 1 0      copy_end   : boolean;
537 876 1 0      units_ready : boolean;

538 878 1 0      begin

538 880 1 1          copy_end := false;
539 881 1 1          repeat

539 883 1 2              clr_screen;
540 884 1 2              writeln;
541 885 1 2              writeln;
542 886 1 2              writeln('      Mount source tape on unit 0, destination tape on unit 1.');


```

543 887 1 2 writeln;
544 888 1 2 writeln(' Hit return when ready, c then return to cancel.');


```

545 889 1 2              writeln;
546 890 1 2              if not eoln then copy_end := true;
548 891 1 2              readln;

549 893 1 2              if not copy_end then begin

550 895 1 3                  Select(source_unit, Tape_data[source_unit]);
551 896 1 3                  Cntrl_status(Tape_status);
552 897 1 3                  units_ready := true;
553 898 1 3                  if S_err in Tape_status then begin

554 900 1 4                      writeln('      The source unit is not ready.');


```

555 901 1 4 writeln;
556 902 1 4 units_ready := false;

557 904 1 4 end(*if*);
559 905 1 3 Rewind_tape;

560 907 1 3 Select(dest_unit, Tape_data[dest_unit]);
561 908 1 3 Cntrl_status(Tape_status);
562 909 1 3 if S_err in Tape_status then begin

563 911 1 4 writeln(' The destination unit is not ready.');


```

564 912 1 4                      writeln;
565 913 1 4                      units_ready := false;

566 915 1 4                  end(*if*);

```


```


```


```


```

SERIES-III Pascal-86, V3.0
TAPE_COPY

16/06/86 PAGE 35
TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC
568  916  1  3          if HWL in Tape_status then begin

569  918  1  4              writeln('    The destination unit is not write enabled.');
```

570 919 1 4 writeln;

571 920 1 4 units_ready := false;

572 922 1 4 end(*if*);

574 923 1 3 Rewind_tape;

575 925 1 3 if units_ready then begin

576 927 1 4 Last_op_EOF := true;

577 928 1 4 count := 0;

578 930 1 4 repeat

578 932 1 5 Select(source_unit, Tape_data[source_unit]);

579 933 1 5 Input_buf.Count_in := 8192;

580 934 1 5 Retry_cnt := Max_retrys;

581 935 1 5 Read_ver;

582 936 1 5 if (Tape_status*Error_bits <> []) then begin

583 938 1 6 writeln(' Fatal read error on source at block ', count);

584 939 1 6 copy_end := true;

585 940 1 6 end(*if*);

586 941 1 5 else if EOF in Tape_status then begin

588 943 1 6 writeln(count, 'blocks written, EOF mark detected.');

589 944 1 6 if (Last_op_EOF and ANSI and (Input_buf.Write_data[0] = 'E'))

or (Last_op_EOF and (not ANSI)) then begin

590 947 1 7 writeln(' End of tape detected, end of copy');

591 948 1 7 copy_end := true;

592 950 1 7 end;

594 952 1 6 Select(dest_unit, Tape_data[dest_unit]);

595 953 1 6 Write_EOF;

596 954 1 6 Last_op_EOF := true;

597 955 1 6 count := 0;

598 957 1 6 end(*if*);

599 959 1 5 else begin

600 961 1 6 Select(dest_unit, Tape_data[dest_unit]);

601 962 1 6 Input_buf.Write_data := Output_buf.Read_data;

602 963 1 6 Input_buf.Count_in := Data_size;

603 964 1 6 Retry_cnt := Max_retrys;

604 965 1 6 Write_ver;

605 966 1 6 if (Tape_status*Error_bits <> []) then begin

606 968 1 7 writeln(' Fatal write error on destination tape.');

607 969 1 7 copy_end := true;

SERIES-III Pascal-86, V3.0
TAPE_COPY

16/06/86 PAGE 36
TAPE_CONTROLLER_MAIN

```
SYMT LINE NESTING    SOURCE TEXT: :F3:ADMTN.SRC
608 971 1 7          end
609 972 1 6          else count := count + 1;
611 973 1 6          Last_op_EOF := false;
612 974 1 6          end;

614 976 1 5          until copy_end;

616 978 1 4          writeln;
617 979 1 4          writeln;
618 980 1 4          write('      Hit return to continue. ');
619 981 1 4          readln;

620 983 1 4          end(*if*);

622 985 1 3          end(*if*);

624 987 1 2          until copy_end;

626 989 1 1          Select(source_unit, Tape_data[source_unit]);
627 990 1 1          Rewind_tape;
628 991 1 1          Select(dest_unit, Tape_data[dest_unit]);
629 992 1 1          Rewind_tape;

630 994 1 1          end;

      *EJECT
```


SERIES-III Pascal-86, V3.0

16/06/86 PAGE 37
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: IF3:ADMTCN.SRC

631 1000 0 0

procedure Local_ops;

(* This procedure allows the user to execute various operations directly on the tape units. All operation which involve data are done on a local buffer. The following operations may be performed :

1. Select a unit.
2. Read a block into the local buffer.
3. Write a block from the local buffer.
4. Write a block with a long inter-record gap.
5. Write a tape mark.
6. Space forward or backward by a specified number of blocks.
7. Set the read threshold either low (normal) or high.
8. Rewind the unit.
9. Set the unit off-line.
10. Display and modify the local buffer.
11. Display diagnostics information on the information stored in the local buffer.
12. Change the size of the local buffer.

*)

632 1033 1 0

const Test_msg = 'TAPE TEST MADRV ';

633 1034 1 0

Test_msg_len = 16;

634 1037 1 0

var Total_count : integer;

635 1038 1 0

Line_count : integer;

636 1039 1 0

Char_count : integer;

637 1040 1 0

i : integer;

638 1041 1 0

Local_ops_exit : boolean;

639 1042 1 0

Cmd_retry : boolean;

640 1043 1 0

Display_buf : packed array [0..15] of char;

641 1044 1 0

Display_exit : boolean;

642 1045 1 0

Parity : boolean;

643 1046 1 0

State_val : integer;

644 1047 1 0

dummy_char : char;

645 1048 1 0

local_ops_msg : Out_string;

646 1049 1 0

local_ops_int : integer;

647 1050 1 0

local_ops_d_en : boolean;

648 1052 1 0

begin

SERIES-III Pascal-86, V3.0
LOCAL_OPS

16/06/86 PAGE 38
TAPE_CONTROLLER_MAIN

SYMT LINE VESTING

SOURCE TEXT: :F3:ADMTN.SRC

```

648 1054 1 1      for i := 0 to Test_msg_len - 1 do Output_buf.Read_data[i] := Test_msg[i] + 1;
650 1055 1 1      Data_size := Test_msg_len;
651 1056 1 1      Local_ops_exit := false;
652 1057 1 1      local_ops_msg := blanks;
653 1058 1 1      local_ops_int := 0;
654 1059 1 1      local_ops_d_en := false;

655 1061 1 1      While not Local_ops_exit do begin

656 1063 1 2          i := Menu(Local_ops_menu, local_ops_d_en, local_ops_msg, local_ops_int);
657 1064 1 2          local_ops_msg := blanks;
658 1065 1 2          local_ops_int := 0;
659 1066 1 2          local_ops_d_en := false;

660 1068 1 2          case i of

661 1070 1 3              1 : Set_unit_num;

662 1072 1 3              2 : begin

                          (* Read a block. *)

662 1076 1 4                  Data_size := 8192;
663 1077 1 4                  Read_block(Output_buf.Read_data, Data_size);
664 1078 1 4                  local_ops_msg := '          Read block size : ' ;
665 1079 1 4                  local_ops_int := Data_size;
666 1080 1 4                  local_ops_d_en := true;

667 1082 1 4                  end;

669 1084 1 3              3 : Write_block(Output_buf.Read_data, Data_size);

670 1086 1 3              4 : Write_with_long_gap(Output_buf.Read_data, Data_size);

671 1088 1 3              5 : Write_eof;

672 1090 1 3              6 : begin

                          (* Space forward. *)

672 1094 1 4                  i := 1;
673 1095 1 4                  Get_value('          Input number of blocks ', i, 0, 30000);
674 1096 1 4                  Space(i);
675 1097 1 4                  local_ops_msg := '          Number of blocks spaced : ' ;
676 1098 1 4                  local_ops_int := i;
677 1099 1 4                  local_ops_d_en := true;

678 1101 1 4                  end;

680 1103 1 3              7 : begin

                          (* Space backward. *)

680 1107 1 4                  i := 1;

```


SERIES-III Pascal-86, V3.0
LOCAL_OPS

16/06/86 PAGE 39
TAPE_CONTROLLER_MAIN

```

SYMT LINE NESTING      SOURCE TEXT: F3:ADMTN.SRC
681 1108 1 4           Get_value('      Input number of blocks ', i, 0, 30000);

682 1110 1 4           i := -i;
683 1111 1 4           Space(i);
684 1112 1 4           local_ops_msg := '      Number of blocks spaced : ';
685 1113 1 4           local_ops_int := i;
686 1114 1 4           local_ops_d_en := true;

687 1116 1 4           end;

689 1118 1 3           8 : Set_RTH_high;

690 1120 1 3           9 : Set_RTH_low;

691 1122 1 3           10 : Rewind_tape;

692 1124 1 3           11 : Off_line;

693 1126 1 3           12 : begin

                        (* Display the local buffer. *)

693 1130 1 4           Display_exit := false;
694 1131 1 4           Total_count := 0;
695 1132 1 4           while not Display_exit do begin

695 1134 1 5               Line_count := 0;
697 1135 1 5               while (not display_exit) and (Line_count < 20) do begin

698 1137 1 6                   if Total_count < Data_size then write(Total_count:4, ' ');
700 1138 1 6                   Char_count := 0;
701 1139 1 6                   while Char_count < 16 do

702 1141 1 6                       if Total_count < data_size then begin

703 1143 1 7                           Disp_hex(Output_buf.Read_data[Total_count]);
704 1144 1 7                           Display_buf[Char_count] :=
                                Ascii_char(Output_buf.Read_data[Total_count]);
705 1146 1 7                           Total_count := Total_count + 1;
706 1147 1 7                           Char_count := Char_count + 1;
707 1148 1 7                           end

708 1150 1 6                       else begin

709 1152 1 7                           write(' ');
710 1153 1 7                           Display_buf[Char_count] := ' ';
711 1154 1 7                           Char_count := Char_count + 1;
712 1155 1 7                           Display_exit := true;
713 1156 1 7                           end(*while*);

715 1158 1 6                       for i := 0 to 15 do write(Display_buf[i]);
717 1159 1 6                       writeln;
718 1160 1 6                       Line_count := Line_count + 1;

719 1162 1 6                   end;

```

SERIES-III Pascal-86, V3.0
LOCAL_OPS

16/06/86 PAGE 40
TAPE_CONTROLLER_MAIN

```

SYMT LINE NESTING      SOURCE TEXT: :F3:ADMTON.BRC

721 1164 1 5           write('Hit return for done');

722 1166 1 5           if not eoln then Display_exit := true;
724 1167 1 5           readln;
725 1168 1 5           end;
727 1169 1 4           end;

729 1171 1 3           13 : begin

                        (* Modify the local buffer. *)

729 1175 1 4           Cmd_retry := false;
730 1176 1 4           repeat

730 1178 1 5               if Cmd_retry then writeln('Number selected not in buffer');
732 1179 1 5               write(' Number of first data byte to modify (0..', Data_size, ')');
733 1180 1 5               Cmd_retry := true;

734 1182 1 5           until Get_int_in_range(i, 0, Data_size);

736 1184 1 4           i := i - 1;
737 1185 1 4           dummy_char := Output_buf.Read_data[i];
738 1186 1 4           repeat

738 1188 1 5               Output_buf.Read_data[i] := dummy_char;
739 1189 1 5               i := i + 1;
740 1190 1 5               dummy_char := Output_buf.Read_data[i];
741 1191 1 5               write(i:4, ' ');
742 1192 1 5               Disp_hex(dummy_char);
743 1193 1 5               dummy_char := Ascii_char(dummy_char);

744 1195 1 5               write(' ', dummy_char, ' ', ' ');
745 1196 1 5               write('/');

746 1198 1 5           until not Get_char_in_range(dummy_char, 0, 255);

748 1200 1 4           end;

750 1202 1 3           14 : begin

                        (* Display diagnostics information. *)

750 1206 1 4           Total_count := 0;
751 1207 1 4           Display_exit := false;
752 1208 1 4           repeat
752 1209 1 5               Line_count := 0;
753 1210 1 5               repeat
753 1211 1 6                   write(Total_count:4, ' ');
754 1212 1 6                   Dummy_char := Output_buf.Read_data[Total_count];
755 1213 1 6                   Disp_hex(dummy_char);
756 1214 1 6                   Diag(Output_buf.Read_data, Total_count, Parity, State_val);
757 1215 1 6                   write(' State = ', State_val);
758 1216 1 6                   if parity then writeln(' Parity error.')
759 1217 1 6                   else writeln;

```


SERIES-III Pascal-86, V3.0
LOCAL_OPS

16/06/86

PAGE 41

TAPE_CONTROLLER_MAIN

```

STMT LINE  NESTING      SOURCE TEXT: :F3:ADMTCN.SRC
761 1218  1  6          Total_count := Total_count + 1;
762 1219  1  6          Line_count := Line_count + 1;
763 1220  1  6          if Total_count > Data_size then Display_exit := true;
765 1221  1  6          until Display_exit or (Line_count = 20);
767 1222  1  5          write('Hit return to continue');
768 1223  1  5          if not eoln then Display_exit := true;
770 1224  1  5          readln;
771 1225  1  5          until Display_exit;
773 1226  1  4          end;

775 1229  1  3          15 : begin

                          (* Change the buffer size. *)

775 1233  1  4          Cmd_retry := false;
776 1234  1  4          repeat

776 1236  1  5          if Cmd_retry then writeln ('Requested buffer size is out of range. ');
778 1237  1  5          Cmd_retry := true;
779 1238  1  5          writeln(' Current buffer size : ', Data_size);
780 1239  1  5          write(' Enter new buffer size : ');

781 1241  1  5          until Get_int_in_range(Data_size, 1, 8192);

783 1243  1  4          end;

785 1245  1  3          16 : Local_ops_exit := true;

786 1247  1  3          end(*case*);

788 1249  1  2          end(*while*);

790 1251  1  1          end;

                          *EJECT

```

SERIES-III Pascal-86, V3.0

16/06/86 PAGE 42
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTCN.SRC

```

791 1257 0 0      procedure System_functions;

(* This procedure is the system function menu. System parameters are set here
as follows :

    1. Data link and terminal baud rates.

    2. Terminal type.

    3. The timing parameters of each tape unit may be changed.

    4. The frame and block diagnostics system may be enabled or disabled.

*)

792 1272 1 0      var    Sys_func_exit    : boolean;

793 1274 1 0      begin

793 1276 1 1          Sys_func_exit := false;

794 1278 1 1          While not Sys_func_exit do case Menu(System_menu, false, blanks, 0) of

796 1280 1 2              1 : Set_baud(0, (Menu(Baud_menu, false, blanks, 0)-1));

797 1282 1 2              2 : Set_baud(1, (Menu(Baud_menu, false, blanks, 0)-1));

798 1284 1 2              3 : Term_type := Menu(Term_menu, false, blanks, 0);

799 1286 1 2              4 : begin

799 1288 1 3                  Set_unit_num;
800 1289 1 3                  Get_value('      Master clock      ',
801 1291 1 3                      Tape_data[Unit].Master_clock, 1, 30000);
802 1293 1 3                  Get_value('      Short gap count   ',
803 1295 1 3                      Tape_data[Unit].Short_gap, 1, 30000);
804 1297 1 3                  Get_value('      Long gap count    ',
805 1299 1 3                      Tape_data[Unit].Long_gap, 1, 30000);
806 1301 1 3                  Get_value('      Normal start gap length ',
807 1304 1 3                      Tape_data[Unit].Start_norm, 1, 30000);
808 1306 1 3                  Get_value('      EOF start gap length  ',
809 1308 1 3                      Tape_data[Unit].Start_norm, 1, 30000);
810 1310 1 3                  Get_value('      Read start gap length ',
811 1312 1 3                      Tape_data[Unit].Start_read, 1, 30000);
812 1314 1 3                  Get_value('      Tape stop gap length ',
813 1316 1 3                      Tape_data[Unit].Stop, 1, 30000);

814 1318 1 3              end;

815 1320 1 2              5 : Block_diag := true;

816 1322 1 2              6 : Block_diag := false;

```

SERIES-III Pascal-86, V3.0
SYSTEM_FUNCTIONS

16/06/86 PAGE 43
TAPE_CONTROLLER_MAIN

```
STMT LINE NESTING    SOURCE TEXT: :F3:ADMTCN.SRC
811 1310 1 2          7 : Frame_diag := true;

812 1312 1 2          8 : Frame_diag := false;

813 1314 1 2          9 : Sys_func_exit := true;

814 1316 1 2          end(*case*);

816 1318 1 1          end;

$EJECT
```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 44

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADMTON.SRC

\$interrupt(Block_rec)

(* This procedure is the remote mode control procedure. It is initiated by the block received interrupt generated by the block IO module. Command blocks consist of the following :

Opcode : byte specifying the required operation. The lower 6 bits are the opcode, bit 6 the ANSI tape format bit and bit 7 the retry suppress bit.

Unit number : byte specifying the logical unit number.

Status : status of the tape unit saved from the previous operation. This is a 16 bit word.

Count value : This is a 16 bit integer specifying either the number of bytes of data in the block transferred, the maximum number of data bytes which may be returned, or the number of blocks or files to be spaced.

Data bytes : A variable number of data bytes to maximum of 8192.

This block is processed and a return block generated. This has the following format :

Status : status of the unit as described above.

Count : count value resulting from the operation as described above. Note that for status operations this contains the unit status in the format same format as the status word.

Return code : this is the PDP QIO return code.

Note that the values returned for count and status are the PDP status words one and two.

*)

817 1362 0 0

procedure Block_rec;

818 1365 1 0

var cmd : integer;

819 1366 1 0

No_retrys : boolean;

820 1367 1 0

Blk_cnt : integer;

821 1368 1 0

Space_dir : integer;

822 1369 1 0

i : integer;

823 1372 1 0

begin

823 1374 1 1

ENABLEINTERRUPTS;

824 1375 1 1

Block_in(Input_buf);

SERIES-III Pascal-86, V3.0
BLOCK_REC

16/06/86 PAGE 45
TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC
825 1376 1 1           cmd := ord(Input_buf.Opcode);
826 1377 1 1           LUN := ord(Input_buf.Unit_num);
827 1378 1 1           Unit := Unit_table[LUN];           (* Set the physical unit number. *)
828 1379 1 1           Output_buf.Tape_status_out := Input_buf.Tape_status_in;
829 1380 1 1           Output_buf.Ret_code := IS_SUC;      (* Assume all OK. *)
830 1381 1 1           Data_size := 0;
831 1382 1 1           Output_buf.Count_out := 0;

832 1385 1 1           if cmd > 127 then begin             (* Check the retry suppress bit. *)
833 1386 1 2               cmd := cmd - 128;
834 1387 1 2               Retry_cnt := 1;
835 1388 1 2               end
836 1389 1 1           else Retry_cnt := Max_retrys;

838 1391 1 1           if cmd > 63 then begin              (* Check the ANSI tape bit. *)
839 1393 1 2               cmd := cmd - 64;
840 1394 1 2               ANSI := true;

841 1396 1 2               end
842 1397 1 1           else ANSI := false;

844 1399 1 1           Select(Unit, Tape_data[Unit]);
845 1400 1 1           Cntrl_status(Tape_status);

846 1402 1 1           if (BOT in Tape_status) or (EOF in Input_buf.Tape_status_in) then Last_op_EOF := true
847 1403 1 1           else Last_op_EOF := false;          (* This is needed for the end of volume check. *)

849 1405 1 1           if Block_diag then begin
850 1406 1 2               writeln('Cmd code = ', cmd);
851 1407 1 2               writeln('Unit number = ', LUN);
852 1408 1 2               Display_status(Input_buf.Tape_status_in);
853 1409 1 2               if ANSI then writeln('ANSI labeled tape');
855 1410 1 2               writeln('Input count = ', Input_buf.Count_in);
856 1411 1 2           end;

858 1414 1 1           if not (S_err in Tape_status) then begin
859 1416 1 2               case cmd of
860 1418 1 3                   RL3_cmd : if Input_buf.Count_in >= 8 then begin

                                   (* Read logical block. *)

                                   Data_size := Input_buf.Count_in;
                                   Read_ver;
                                   Set_code;
                                   Ver_check;
                                   if (Data_size < 8) and not (EOF in Tape_status) then
                                   Output_buf.Ret_code := IE_BBE;

                                   Output_buf.Count_out := Data_size;
                                   if Output_buf.Ret_code = IE_EOT then Output_buf.Ret_code := IS_SUC;

```


SERIES-III Pascal-86, V3.0
BLOCK_REC

16/06/86 PAGE 46
TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:A0MTCN.SRC

```

870 1432 1 4      if Output_buf.Ret_code = IE_EOF then Output_buf.Count_out := 2;
872 1433 1 4      end

873 1435 1 3      else Output_buf.Ret_code := IE_SPC;

875 1437 1 3      WLB_cmd : if not ((HWL in Tape_status) or (SWL in Input_buf.Tape_status_in)) then
876 1438 1 3          if (Input_buf.Count_in >= 14) and (Input_buf.Count_in <= 8192) then
877 1439 1 3              begin

                  (* Write logical block. *)

877 1443 1 4          Data_size := Input_buf.Count_in;

878 1445 1 4          Write_ver;
879 1446 1 4          Set_code;
880 1447 1 4          Ver_check;
881 1448 1 4          Output_buf.Count_out := Data_size;
882 1449 1 4          if Data_size < 14 then Output_buf.Ret_code := IE_BBE;
884 1450 1 4          Data_size := 0;

885 1452 1 4          end

886 1454 1 3          else Output_buf.Ret_code := IE_SPC
887 1455 1 3          else Output_buf.Ret_code := IE_WLK;

889 1457 1 3      EOF_cmd : if not ((HWL in Tape_status) or (SWL in Input_buf.Tape_status_in)) then
890 1458 1 3          begin

                  (* Write a tape mark. *)

890 1462 1 4          Write_EOF;
891 1463 1 4          Cntrl_status(Tape_status);
892 1464 1 4          if (Tape_status#Error_bits <> 0) then Output_buf.Ret_code :=
893 1465 1 4              IE_FHE;
894 1466 1 4          Tape_status := Tape_status + [EOF];

895 1468 1 4          end

896 1470 1 3          else Output_buf.Ret_code := IE_WLK;

898 1473 1 3      SPB_cmd : begin

                  (* Space blocks. *)

898 1477 1 4          Output_buf.Count_out := Input_buf.Count_in;
899 1478 1 4          Space(Output_buf.Count_out);
900 1479 1 4          Cntrl_status(Tape_status);
901 1480 1 4          Set_code;
902 1481 1 4          if (Input_buf.Count_in > 0) and (Output_buf.Count_out = 0) then
903 1482 1 4              EDV_test;

904 1484 1 4          Data_size := 0;

```

SERIES-III Pascal-86, V3.0
BLOCK_REC

16/06/86 PAGE 47
TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC
905 1486 1 4           Output_buf.Count_out := abs(Output_buf.Count_out);

906 1488 1 4           end;

908 1490 1 3           SPF_cmd : begin

                        (* Space files. *)

908 1494 1 4           if Input_buf.Count_in > 0 then Space_dir := 1
909 1495 1 4           else Space_dir := -1;

911 1497 1 4           Output_buf.Count_out := Input_buf.Count_in;

912 1499 1 4           while (Output_buf.Count_out <> 0) and (not (EOV in Tape_status)) do
913 1500 1 4           begin

913 1502 1 5               Blk_cnt := MAXINT*Space_dir;
914 1503 1 5               Space(Blk_cnt);
915 1504 1 5               Cntrl_status(Tape_status);
916 1505 1 5               if (Space_dir = 1) and (Blk_cnt = 0) then EOV_test;
918 1506 1 5               Last_op_EOF := true;
919 1507 1 5               Output_buf.Count_out := Output_buf.Count_out - Space_dir;
920 1508 1 5               if EOT in Tape_status then Output_buf.Ret_code := IE_EOF;

922 1510 1 5           end;

924 1512 1 4           Output_buf.Count_out := Input_buf.Count_in - Output_buf.Count_out;

925 1514 1 4           Data_size := 0;

926 1516 1 4           Output_buf.Count_out := abs(Output_buf.Count_out);

927 1518 1 4           end;

929 1521 1 3           RWD_cmd : begin

929 1523 1 4               Rewind_tape;
930 1524 1 4               Cntrl_status(Tape_status);

931 1526 1 4           end;

933 1529 1 3           RWU_cmd : begin

933 1531 1 4               Rewind_tape;
934 1532 1 4               Off_line;
935 1533 1 4               Cntrl_status(Tape_status);

936 1536 1 4           end;

938 1538 1 3           STC_cmd : begin

                        (* Set status. Note that the required status is passed

```


SERIES-III Pascal-86, V3.0
BLOCK_REC

16/06/86 PAGE 48
TAPE_CONTROLLER_MAIN

```

SYNT LINE NESTING      SOURCE TEXT: :F3:ADMTN.SRC
                                and returned in the count field. *)

938 1543 1 4            Input_buf.Write_block[2] := Input_buf.Write_block[4];
939 1544 1 4            Input_buf.Write_block[3] := Input_buf.Write_block[5];

940 1546 1 4            Output_buf.Tape_status_out := Output_buf.Tape_status_out
                                *Change_bits + Input_buf.Tape_status_in*Status_bits;

941 1549 1 4            Output_buf.Read_block[2] := Output_buf.Read_block[0];
942 1550 1 4            Output_buf.Read_block[3] := Output_buf.Read_block[1];

943 1552 1 4            Tape_status := Tape_status + [EOF, EOV]*Input_buf.Tape_status_in;

944 1554 1 4            end;

946 1556 1 3            SMO_cmd : begin

                                (* Set characteristics and mount tape. This is the same
                                STC except that the operation is rejected if the unit
                                is not at BOT and ready. *)

946 1562 1 4            if not (BOT in Tape_status) then Output_buf.Ret_code := IE_FHE;

948 1564 1 4            Input_buf.Write_block[2] := Input_buf.Write_block[4];
949 1565 1 4            Input_buf.Write_block[3] := Input_buf.Write_block[5];

950 1567 1 4            Output_buf.Tape_status_out := Output_buf.Tape_status_out
                                *Change_bits + Input_buf.Tape_status_in*Status_bits;

951 1570 1 4            Output_buf.Read_block[2] := Output_buf.Read_block[0];
952 1571 1 4            Output_buf.Read_block[3] := Output_buf.Read_block[1];

953 1573 1 4            Tape_status := Tape_status + [EOF, EOV]*Input_buf.Tape_status_in;

954 1575 1 4            end;

956 1577 1 3            SEC_cmd : begin

                                (* Sense characteristics. They are passed in the count
                                field. *)

956 1582 1 4            Output_buf.Read_block[2] := Output_buf.Read_block[0];
957 1583 1 4            Output_buf.Read_block[3] := Output_buf.Read_block[1];

958 1585 1 4            Tape_status := Tape_status + [EOF, EOV]*Input_buf.Tape_status_in;

959 1587 1 4            end;

961 1589 1 3            otherwise Output_buf.Ret_code := IE_IFC;

963 1591 1 3            end;

965 1593 1 2            end

966 1595 1 1            else Output_buf.Ret_code := IE_FHE;

```

SERIES-III Pascal-86, V3.0
BLOCK_REC

16/06/86 PAGE 49
TAPE_CONTROLLER_MAIN

STMT LINE NESTING SOURCE TEXT: :F3:ADMTCN.SRC

968 1598 1 1 Output_buf.Tape_status_out := Input_buf.Tape_status_in*Hold_bits + Tape_status*Change_bits;

969 1601 1 1 if Block_diag then writeln('Count out = ',Output_buf.Count_out);

971 1603 1 1 Block_out(Output_buf,Data_size + 6);

972 1605 1 1 end;

\$EJECT

SERIES-III Pascal-86, V3.0

16/06/86

PAGE 50

TAPE_CONTROLLER_MAIN

STMT LINE NESTING

SOURCE TEXT: :F3:ADNTCN.SRC

```
(* This is the main program. The controller is initialized, a message is output
to the screen and the system waits for either the unit to be set to local mode
or a receive interrupt. *)
```

```
973 1616 0 0      begin

973 1618 0 1      Tape_data[0].Master_clock := Unit_0_MC;
974 1619 0 1      Tape_data[0].Short_gap := 2 * Unit_0_MC;
975 1620 0 1      Tape_data[0].Long_gap := 12 * Unit_0_MC;
976 1621 0 1      Tape_data[0].Start_norm := 900;
977 1622 0 1      Tape_data[0].Start_long := 3000;
978 1623 0 1      Tape_data[0].Start_EOF := 3000;
979 1624 0 1      Tape_data[0].Start_read := 200;
980 1625 0 1      Tape_data[0].Stop := 88;

981 1627 0 1      Tape_data[1].Master_clock := Unit_1_MC;
982 1628 0 1      Tape_data[1].Short_gap := 2 * Unit_1_MC;
983 1629 0 1      Tape_data[1].Long_gap := 12 * Unit_1_MC;
984 1630 0 1      Tape_data[1].Start_norm := 900;
985 1631 0 1      Tape_data[1].Start_long := 3000;
986 1632 0 1      Tape_data[1].Start_EOF := 3000;
987 1633 0 1      Tape_data[1].Start_read := 200;
988 1634 0 1      Tape_data[1].Stop := 88;

989 1636 0 1      Hold_bits := [P_err, CRC_err, DMA_err, SWL, 16X, xx];
990 1637 0 1      Error_bits := [P_err, CRC_err, Length_err, DMA_err, Time_err];
991 1638 0 1      Change_bits := [EOT, EOF, S_err, RWD, HWL, BOT, EDV];
992 1639 0 1      Status_bits := [P_err, CRC_err, DMA_err, SWL, 16X, xx];
993 1640 0 1      Code_errors := [P_err, CRC_err];

994 1642 0 1      ANSI := true;

995 1644 0 1      SETINTERRUPT(Block_rec_int, Block_rec);

996 1646 0 1      Unit_table[0] := 1;
997 1647 0 1      Unit_table[1] := 0;

998 1650 0 1      LUN := 0;
999 1651 0 1      Unit := Unit_table[LUN];
1000 1652 0 1      Block_diag := false;
1001 1653 0 1      Term_type := 1;

1002 1655 0 1      Frame_diag := false;

1003 1658 0 1      Init_tape;
1004 1659 0 1      Init_block_IO;

1005 1662 0 1      ENABLEINTERRUPTS;
```


SERIES-III Pascal-86, V3.0

16/06/86

PAGE 31

TAPE_CONTROLLER_MAIN

```

STMT LINE NESTING      SOURCE TEXT: :F3:ADMTCN.SRC
1005 1664 0 1          clr_screen;
1007 1665 0 1          writeln;
1008 1666 0 1          writeln;
1009 1667 0 1          writeln('          SERIAL TAPE CONTROLLER');
1010 1668 0 1          writeln('          To enter local mode hit return');
1011 1669 0 1          readln;          (* The unit waits here for a CR on the input. *)

1012 1671 0 1          DISABLEINTERRUPTS;

1013 1673 0 1          while true do case Menu(Main_menu, false, blanks, 0) of

1015 1675 0 2              1 : begin
1015 1676 0 3                  ANSI := false;
1016 1677 0 3                  Tape_copy;
1017 1678 0 3              end;

1019 1680 0 2              2 : begin
1019 1681 0 3                  ANSI := true;
1020 1682 0 3                  Tape_copy;
1021 1683 0 3              end;

1023 1685 0 2              3 : Wipe_tape;

1024 1687 0 2              4 : System_functions;

1025 1689 0 2              5 : Local_ops;

1026 1691 0 2              6 : begin
1026 1692 0 3                  writeln('          To return to local mode hit return');
1027 1693 0 3                  ENABLEINTERRUPTS;
1028 1694 0 3                  Init_block_IO;
1029 1695 0 3                  readln;          (* Go wait for the next local mode request. *)
1030 1696 0 3                  DISABLEINTERRUPTS;
1031 1697 0 3              end;

1033 1699 0 2          end (*case*);

1035 1701 0 1          end.

```

Summary Information:

PROCEDURE	OFFSET	CODE SIZE	DATA SIZE	STACK SIZE
DISPLAY_STATUS	0BF4H	0223H	547D	000EH 140
CAP	0E17H	002EH	46D	0006H 60
VAL	0E45H	003BH	59D	000AH 100
IN_INT	0E80H	029EH	670D	0046H 700
GET_INT_IN_RANGE	111EH	0032H	500	000CH 120
GET_CHAR_IN_RANGE	1150H	0040H	640	0012H 180
GET_VALUE	1190H	00CAH	202D	0010H 160
CLR_SCREEN	125AH	00E8H	232D	0014H 200
MENU	1342H	00DEH	222D	0016H 220
MAIN_MENU	1420H	0132H	306D	0010H 160

SERIES-III Pascal-86, V3.0

Summary Information

16/06/86

PAGE 52

TAPE_CONTROLLER_MAIN

SYSTEM_MENU	1552H	0198H	4110	0010H	160
TERM_MENU	16EDH	00E3H	2270	0010H	160
BAUD_MENU	17D0H	0184H	3680	0012H	180
LOCAL_OPS_MENU	1954H	0209H	5210	0010H	160
HEX_DIGIT_OUT	1B50H	0040H	640	000AH	100
DISP_HEX	1B9DH	0041H	650	000AH	100
ASCII_CHAR	1BDEH	0028H	400	0006H	60
SET_UNIT_NUM	1C06H	00AEH	1740	001CH	280
BS_READ	1CB4H	0052H	820	0012H	180
WRITE_VER	1D06H	00ACH	1720	0012H	180
READ_VER	1DB2H	0049H	730	0010H	160
SET_CODE	1DFBH	002AH	420	0006H	60
VER_CHECK	1E25H	002BH	430	0006H	60
EOV_TEST	1E50H	0047H	710	000EH	140
WIPE_TAPE	1E97H	0148H	3280	0010H	160
TAPE_COPY	1FDFH	0379H	8890	001EH	300
LOCAL_OPS	2358H	05E0H	15040	007AH	1220
SYSTEM_FUNCTIONS	2938H	0260H	6080	0034H	520
BLOCK_REC	2B98H	04C5H	12210	0034H	520
TAPE_CONTROLLER_MAIN	305DH	022EH	5580	28FEH 10494D 002CH	440
-CONST IN CODE-		0BF4H	30600		

Total	328BH	12939D	28FEH 10494D	0302H	7700
-------	-------	--------	--------------	-------	------

1841 Lines Read.

0 Errors Detected.

Dictionary Summary:

120KB Memory Available.

20KB Memory Used (16%).

0KB Disk Space Used.

9KB out of 16KB Static Space Used (56%).

APPENDIX G

SOFTWARE BUILD INFORMATION

1. The PDP-11 device driver. The following series of command are used to build the device driver. Note that the user's terminal must be privileged.

UIC 100,4

DL1:

MAC

MATBL,MATBL/-SP=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:[100,34]MATBL
MAC

MADRV,MADRV/-SP=LB:[1,1]EXEMC/ML,LB:[11,10]RSXMC,SY:[100,34]MADRV
UIC 1,54

TKB

DLO:MADRV/-HD/-MM,,DLO:MADRV=
SY:[100,34]MADRV,SY:[100,34]MATBL

LB:RSX11M.STB/SS

LB:[1,1]EXELIB/LB

/

STACK=0

PAR=GEN:120000:4000

//

2. Tape controller software. A currently configured, the tape controller makes use of a downloaded program, but the software is designed to be programmed into EPROM. The sequence of commands used both to build the downloaded program as well as the Romable version are given below :

2.1. The following command are entered to build the downloadable version. Note that this sequence builds a HEX file which must be down loaded to the tape controller module. Note also that in order to fit both the program and the data into RAM, the main buffers have been reduces in size to 5000 bytes. As the PDP-11 never generates blocks of greater than 2048 characters this has no adverse effect.

```
run pasc86 :f3:admtcn.src
run pasc86 :f3:admbcm.src
run asm86 :f3:admpio.asm
run asm86 :f3:admtio.asm
run asm86 :f3:admcom.asm
run asm86 :f3:admil8.asm
run link86 :f3:admtcn.obj, :f3:admbcm.obj, p86rn0.lib, &
:f3:admtio.obj, :f3:admcom.obj, p86rnl.lib, :F3:admpio.obj, &
rtnull.lib, 87null.lib ma pl sb ty
run loc86 :f3:admtcn.lnk ma pl sb
run :F2:oh86 :f3:admtcn
```

2.2. The following sequence of commands is used to build the version for programming into EPROM. The start address of the EPROM is dependant on the EPROMs used, and it's SEGMENT address is represented here as XXXXH. The EPROM block is always contiguous, and ends at FFFFFH..

```
run pasc86 :f3:admtcn.src
run pasc86 :f3:admbcm.src
run asm86 :f3:admpio.asm
```

```
run asm86 :f3:admtio.asm
run asm86 :f3:admcom.asm
run asm86 :f3:admil8.asm
run link86 AS(Main(XXXX6H) :f3:admil8.obj, :f3:admtcn.obj,
:f3:admbcm.obj, p86rn0.lib, :f3:admtio.obj, :f3:admcom.obj,
p86rnl.lib, :F3:admpio.obj, rtnull.lib, 87null.lib ma pl sb ty
run loc86 AD(CS(CODE(XXXX0H)) IC(XXXX0H) :f3:admtcn.lnk ma pl sb
run :F2:oh86 :f3:admtcn
```


APPENDIX H

PASCAL-86 V3 FAULTS

During this project two serious fault were discovered in the Pascal-86 compiler :

1. Under the Large memory model invalid object records are produced. The compiler produces code on the assumption that all segments will start on a paragraph boundary, but produces object files specifying byte aligned segments. This has no effect when a program is run on the host, as memory can only be allocated on a paragraph boundary. However, when used on a target system, after linkage and location, the program will fail.
2. The run-time system executes an undocumented call to the memory allocation routine (TQALLOCATE) while initializing. If a 48 byte segment is not allocated the program will crash.