OPERATING SYSTEM INTERFACE GUIDE

THE CORVUS CONCEPT

CORVUS SYSTEMS

OPERATING SYSTEM INTERFACE GUIDE

THE CORVUS CONCEPT

PART NO.: 7100-03290

DOCUMENT NO.: CCC/36-99/1.1

RELEASE DATE: February, 1983

TABLE OF CONTENTS

Operating System Interface Guide

Driver Examples

MERLIN OPERATING SYSTEM

Interface Guide

First Edition

4th October 1981

Silicon Valley Software Incorporated
10340 Phar Lap Drive
Cupertino
California 95014

These MERLIN Operating System Manuals were produced by:

Jeffrey Barth, R. Steven Glanville and Henry McGilton.

Silicon Valley Software Incorporated
Publication Number 810830-01

Table of Contents

Chapter 1 Introduction 1

1.1 Overview and Layout of this Guide 2

Chapter 2 General Information 3

2.1 Units 3
2.2 Data Representations in MERLIN 4

2.2.1 Characters, Words, and Long Words 4
2.2.2 Boolean Data Type 5
2.2.3 The NIL Pointer 5
2.2.4 The String Data Type 5
2.2.5 Packed Array of Character 5

2.3 The System Communication Area 7
2.4 The System Call Vector 9

2.4.1 Calling a System Routine 11

2.5 File Information Block (FIB) 11
2.6 Device Directory 15

2.6.1 Directory Entry for a Header Record 15
2.6.2 Directory Entry for a File Entry 16

2.7 The Device or Unit Table 18
2.8 Input Output Result Codes 20
2.9 Memory Layout under MERLIN on the 68000 22
2.10 Register Usage in MERLIN 23
2.11 Environment of A Running Program 23

Chapter 3 System Calls 25

3.1 Unit input-output 25

3.1.1 UNITREAD and UNITWRITE - Direct Unit Data Transfer 25
3.1.2 UNITBUSY - Check if Unit is Busy 26
3.1.3 UNITCLEAR - Reset a Unit 26
3.1.4 UNITSTATUS - Return Status of Unit 26

3.2 File input-output 27

3.2.1 FINIT - Initialize a File 27
3.2.2 FGET and FPUT - Transfer File Data 28
3.2.3 FOPEN - Open File 28
3.2.4 FCLOSE - Close File 28
3.2.5 READCHAR - Read a Character from a File 29
3.2.6 WRITECHAR - Write a Character to a File 29
3.2.7 SEEK - Position to a Specific Record in a File 29
3.2.8 BLOCKIO - Block input-output 30

3.3 Memory Management 31

3.3.1 NEW - Allocate Storage 31
3.3.2 DISPOSE - De-Allocate Storage 31
3.3.3 MARK and RELEASE - Mark Heap and Release Heap 32
3.3.4 MEMAVAIL - Determine Available Memory 32

3.4 GETDIR - Read a Directory 32

Chapter 4 Writing a Unit Driver 34

4.1 Calling Conventions 34

4.1.1 Unit Driver Command Parameter 35

4.2 A Sample Unit Driver 36

Chapter 5 Interface Definitions in Pascal 39

5.1 Basic Constant and Type Definitions 39

5.1.1 Layout of the Date Record 41
5.1.2 Layout of a Directory Entry 41
5.1.3 File Interface Block Definition 43
5.1.4 System Communication Area Definition 44
5.1.5 Layout of the Device Table 44
5.1.6 Layout of the Process Table 44

5.2 Procedure Interfaces in PASCAL 46

5.2.1 Unit Input Output 46
5.2.2 File Input Output 47
5.2.3 Memory Management 48
5.2.4 Miscellaneous Procedures 48

PREFACE

MERLIN is a "mini" operating system for computer systems based on the Motorola MC68000 microprocessor.

The MERLIN Operating System Documentation is arranged into two distinct books.

The User's Guide is a "concepts and facilities" manual which explains the core ideas of MERLIN - its command interpreter, file system, and the utility commands that provide a means to get started on MERLIN. The User's Guide also contains information about the software packages and utilities that run under MERLIN. There are descriptions of how to run the compilers, the linker and librarian, and a summary of ED - the line-oriented editor.

The Internals Guide is a MERLIN Internal Interface Guide for programmers wishing to write software to run under MERLIN - it covers topics such as file structures, memory layout, device drivers, and other information about MERLIN.

There are other manuals in addition to these two. The additional manuals are whole, self-contained manuals such as the Pascal and FORTRAN reference manuals. These are separate because (a) they are large and placing them in the User's Guide would make that manual impossibly large, and (b) because they are separately priced-products.

Chapter 1

Introduction

MERLIN is a basic executive program for 68000-based microcomputer systems. Its main purpose is to provide an operating environment in which users can develop and run software applications quickly and easily. MERLIN's main features include:

- Single-user system - the user has the full power and responsiveness of the MC68000 system available with no competition for resources with other users.
- Fixed and demountable volumes (devices).
- Two level file structure.
- UNIX-like command language with re-direction of input and output.
- Automatic startup command file for initialization.
- The shell or command interpreter is simply a system command - users can develop their own shells to suit their specific needs.
- Assignable device drivers - new device drivers can be incorporated without the need for system reconfiguration.

Users view MERLIN as composed of several distinct parts:

- the <u>file system</u> provides a way to store data in named collections called <u>files</u> and a way to create, examine, remove, copy, and otherwise manipulate such files.
- the <u>command interpreter</u>, known as "the shell", provides the basic means of telling MERLIN what things it should do.
- the <u>programming languages</u> provide the means to write new software applications. MERLIN supports Pascal, FORTRAN, an

Assembler, and a Linker.

- the utility software supplies tools to aid in using the system. Utilities include an editor for creating and changing text files, disk-file manipulation programs and object code mangagement programs.

Users use these basic operating system facilities to generate their own applications packages or to do other useful work. There are many commercially available packages for business and scientific use written in languages supported by MERLIN.

On the surface, MERLIN looks somewhat like UNIX (for users familiar with UNIX), in that MERLIN uses the same command layouts and methods to indicate options. MERLIN also uses the same notation for describing files. It should be noted, however, that MERLIN is not UNIX, and does not have the power and capabilities of a full UNIX system.

This document is intended as an internal interface guide for those wishing to write software to run under MERLIN's control.

Internally, MERLIN's file system is not a proper hierarchical file system. The file system in fact is at this time compatible with the UCSD PASCAL file system. There were some good reasons for doing this, the major one being a portability issue.

1.1 Overview and Layout of this Guide

Chapter 2 is a "general information" chapter which describes the basic details of MERLIN, discusses the idea of units, and describes some of the data structures necessary.

Chapter 3 is a detailed breakdown of the various system calls that MERLIN provides.

Chapter 4 provides a description on "how to write a device driver". An annotated sample device is provided.

Chapter 5 is a list of the Pascal types and procedure interfaces that are described in narrative form in Chapter 3. These Pascal interfaces are there for those who are more comfortable in Pascal.

Chapter 2

General Information

This Chapter supplies general information about data structures and the means by which software makes MERLIN system calls. Topics covered in this Chapter are:

- a description of the units that MERLIN supports.
- data representation.
- various data structures such as the system communication area.
- memory layout, and program environment.

2.1 Units

MERLIN, as stated previously, looks somewhat like the UCSD Pascal system. MERLIN knows about several units, that is, external devices to or from which data may be transferred.

Generally speaking, it is only neccessary to be concerned with units when using unit input-output - the software layer below that of file input-output. The unit numbers that MERLIN currently deals with are as follows:

Unit Number and Name	Description
0 - /null	is a "null" device. It acts as an infinite sink or "black hole" when it is written to; when is is read from, an end-of-file condition is returned.
1 - /console	is the console, that is, the keyboard and screen, with echo.

2 - **/systerm**	is the console, that is, the keyboard and screen, without echo.
3 -	is user assignable.
4 -	is the boot disk - the disk from which MERLIN boots up, and the default disk on which MERLIN looks for commands.
5 -	is a user disk. Note that devices 9..12 are also user disks.
6 - **/printer**	is the printer if one is attached.
7 - **/remin**	is the remote input device, such as a serial line.
8 - **/remout**	is the remote output device, corresponding to (7) above.
9 .. 12	are user disks.
13..20	are user assignable devices. There may be different numbers of user assignable devices in different implementations of MERLIN.

2.2 Data Representations in MERLIN

This Section describes the way that data is represented internally in MERLIN.

MERLIN is implemented almost entirely in 68000 Pascal, with a small number of lines of assembler code to perform raw device handling. Thus the discussion on data representations and memory layout represent the Pascal implementations. These notes are for users wishing to interface foreign language implementations to the Pascal oriented MERLIN system.

2.2.1 Characters, Words, and Long Words

Characters, words, and long words are the three basic data types. Data elements which occupy words are always aligned on word (even byte) boundaries.

Characters, or bytes, occupy 16 bits if they are not packed. Packed characters occupy a byte and are aligned on a byte boundary.

Words occupy two bytes, or 16 bits. Words are the Pascal **integer** data types. Words are always aligned on a two byte boundary. Words represent signed integers in the range -32768 .. +32767.

Long Words occupy four bytes, or 32 bits. Long words are always aligned on a two byte boundary. Long words are accessible in Pascal by the **longint** data type. Long words represent signed integers in the range -2,147,483,648 .. +2,147,483,647. Long words are also used to store memory addresses and pointers in Pascal.

2.2.2 Boolean Data Type

The Pascal implementation has a **Boolean** data type. A **Boolean** is always represented in a single byte quantity. A value of 0 (zero) represents **false**. A value of 1 (one) represents **true**. No other values are valid. When a **Boolean** value is not an element of a packed data structure, a full byte of storage is used to facilitate access.

2.2.3 The NIL Pointer

As mentioned above, the Pascal implementation uses a long word or 32-bit quantity to represent a pointer. One of the important pointers is the nil pointer which points to no data element (for example, used to indicate the end of a list). In this implementation, nil is represented by the value zero (0).

2.2.4 The String Data Type

Pascal has a dynamic sized string data type similar to that of the UCSD Pascal system. A string is a sequence of bytes in memory, with the first byte in the string containing the length of the string (not including the first byte). This means that the maximum string length is 255 bytes. A string value must be aligned on a word boundary.

2.2.5 Packed Array of Character

The **Packed Array of Char**(acter) data type is not the same as the length delimited string type described above. The Packed Array of Character is simply a stream of bytes in memory. There is no length field as in the string data type above. As with dynamic sized strings, a packed array of character must be aligned on a word boundary.

2.3 The System Communication Area

MERLIN maintains a System Communication Area in RAM. The System Communication Area contains global information that is important to running programs. Two of the important items are the "IORESULT", which is the return code from input-output operations, and the start address of the system call jump vector.

The System Communication Area base address is contained in the long word found in absolute location $180. The System Communication Area layout is described here.

IORESULT	is a word value which contains a result code after completion of any input-output process.
PROCESS NUMBER	is a word value, which is the current process number. The initial shell is assigned process number 0. Each subsequent process receives an incremented process number.
FREE HEAP	is a long word pointer to the start of the free memory available for storage allocation.
SYSTEM CALL VECTOR	is a long word pointer to the start of the system call vector. The system call vector is a table of jump addresses to the system routines. This is described in more detail later on.
SYSOUT	is a long word pointer to the initial shell's standard output file. SYSIN and SYSOUT are used for court of last resort error messages when the Pascal system runs into trouble, for example, when it runs short of allocatable storage.
SYSIN	is a long word pointer to the initial shell's standard input file.
SYSTEM DEVICE TABLE	is a long word pointer to the device table.
DIRECTORY NAME	is a long word pointer to the currently "logged" directory name.

USER TABLE	is a long word pointer to the start address of the user command table.
DATE RECORD	is the encoded form of the current date. The Date Record occupies one word.
OVERLAY TABLE ADDRESS	is a long word value which is the start address of the overlay table. This value is only used when the running process contains overlays. Otherwise it contains a zero (0).
NEXT PROCESS NUMBER	is a word value that the next process number will be assigned.
NUMBER OF PROCESSES	is a word value representing the number of processes currently active (including the first level shell).
PROCESS TABLE ADDRESS	is a long word pointer to the process table. The process table is simply a save area for process context information.
BOOT NAME	is a long word pointer to the name of the device from which to boot the system.
MEM MAP	is a long word pointer to a table describing the limits of memory available to MERLIN on the current hardware.
BOOTDEV	is a word value representing the device number of the initial boot device.

byte +0	IORESULT
+2	Process Number
+4	Pointer to next available free space on the heap
+8	Pointer to start of System Call Vector
+12	Pointer to System Output File
+16	Pointer to System Input File
+20	Pointer to System Device Table
+24	Pointer to Boot Device Directory Name
+28	Pointer to Start of User Command Table
+32	Today's Date (held as a Packed Record)
+34	Overlay Jump Table Address
+38	Next Process Number
+40	Number of Processes
+42	Pointer to the Process Table Array
+46	Pointer to the Name of the Boot Device
+50	Pointer to Memory Bounds Map
+54	Boot Device Number

Figure 2-1
System Communication Area Layout

2.4 The System Call Vector

All MERLIN system calls are, at this time, made by reference through a vector of procedure addresses. The start address of the system call vector is found in the system communication area, described previously. Each entry in the system call vector is a long word (32-bit) pointer. The table below is a list of the entries in the system call vector.

Offset	Routine Name	Description
0	UNIT WRITE	Direct write to unit.
1	UNIT READ	Direct read from a unit.
2	UNIT CLEAR	Clear - reset a unit.
3	UNIT BUSY	Check if unit is busy.
4	FPUT	Write one record to a file.
5	FGET	Read one record from a file.
6	FINIT	Initialize a file.
7	FOPEN	Open a file.
8	FCLOSE	Close a file.
9	WRITECHAR	Write a character to a file.
10	READCHAR	Read a character from a file.
11	BLOCKIO	Block input-output. Transfer a specified number of blocks to or from a file.
12	FSEEK	Position a file to a specific record.
13	NEW	Allocate memory on the heap.
14	DISPOSE	Remove allocated memory. DISPOSE is currently a no-op. Mmemory management is handled with MARK and RELEASE.
15	MARK	Mark the current position of the top of heap.
16	RELEASE	Cut the heap back to a previously MARK'ed position.
17	MEMAVAIL	Determine amount of memory available for dynamic storage allocation.
18		Get directory name.
19	LOAD1	Calls the loader to load an overlay.
20	REMOVE1	Remove (unload) an overlay.
21	SYSTEM DEBUG	If NIL, there is no debug available.
22	MERLIN	The entry point to restart MERLIN.

Figure 2-2
The System Call Vector

The last four entries in the table above are used by MERLIN and need not normally be accessed by user programs.

2.4.1 Calling a System Routine

To call a system routine, the appropriate parameters must be pushed onto the stack. The last thing pushed onto the stack should be the return address (normally pushed via a JSR instruction). The address of a system routine is extracted from the system-call vector, and a JSR to that address is then executed.

The code fragment below illustrates a way to call a system routine. In this specific example, the routine FCLOSE is called to close a file.

```
PEA         FBUFF         ;  Push address of FIB.
CLR.W       -(SP)         ;  Close type := NORMAL.
MOVE.L      $180.W,A0     ;  A0 := System Communication Area address.
MOVE.L      8(A0),A0      ;  A0 := System Call Vector address.
MOVE.L      32(A0),A0     ;  A0 := Address of FCLOSE entry.
JSR         (A0)          ;  Call the FCLOSE routine.
... Return Address ...;      FCLOSE returns to here
```

2.5 File Information Block (FIB)

Access to files requires passing the address of a File Information Block, abbreviated to FIB. A FIB contains all information about a file, its type, buffering and so on.

Before a file can be opened, an FIB must be allocated. The total number of bytes to be allocated depends on whether using Block input-output is being used. If Block input-outputis being used, the FIB is 64 bytes long. In this case, the user must also allocate a buffer for the block. If Block input-output is not being used, in other words the file is a text file or an ISO file of type, the FIB is 576 bytes long, plus the number of bytes in a record.

WINDOW	is a long word pointer to the file 'window' - the area at the end of the FIB that holds the current record.
END OF LINE	is a Boolean that is true if an end-of-line was encountered in the file, false otherwise.

END OF FILE	is a **Boolean** that is **true** if the file is positioned at end-of-file, false otherwise.
TEXT	is a **Boolean** that is **true** if this is a text file. This is true for interactive (mode 0) or text (mode -2) files. It is false for any other file type.
STATE	is a word value that can take on the values 0, 1, 2 or 3. This field is only used for text files.
RECORD SIZE	is a word quantity that defines the number of bytes in a record.
FILE IS OPEN	is a **Boolean** quantity. When true, the following information in the structure is valid.
FILE IS BLOCKED	is a **Boolean** value that is **true** if the file resides on a blocked device.
UNIT NUMBER	is a word that contains the current unit number for this file.
VOLUME NAME	is an eight byte string that contains the name of the volume on which this file resides. The first byte in the string is the number of bytes in the volume name.
REPEAT COUNT	is a word quantity that represents the number of leading spaces on a line. It is included here for UCSD Pascal compatibility.
NEXT BLOCK	is a word quantity which is the number of the next block to be read from or written to the file. This field only applies when the file is an ISO or a text file.
MAXIMUM BLOCK	is a word quantity that is the number of the last block in the file.
MODIFIED	is a **Boolean** quantity that, when true, indicates that this file has been changed.
HEADER	is a directory entry. This information is used by the file system and contains information such as the file's name, relative disk location and latest modification date. The directory entry

occupies 26 bytes in the FIB.

SOFT BUFFER	is a **Boolean** quantity that when true, indicates that the file buffer for this file is actually a part of this structure, instead of separately allocated as in the case of a blocked file. When SOFT BUFFER is true, the following items are part of the File Information Block.
NEXT BYTE	is a word quantity that is the next byte position to be read or written in the buffer.
MAXIMUM BYTE	is a word quantity that is the number of the last byte in the buffer. This is used when reading a file that has a partial last block or when writing any file.
BUFFER CHANGED	is a **Boolean** quantity that when true, indicates that the file buffer in this FIB has been changed and therefore must be eventually written back to the disk.
BUFFER	is a 512 byte array - the size of one logical disk block.
RECORD WINDOW	is an array of bytes sufficiently large to hold one record from the file. If that record is an odd number of bytes in size, the buffer is increased to be an even number of bytes long.

The diagram on the next page is a graphic layout of a File Information Block.

```
--------> +----------------------------------------+
byte +0   |        Pointer to the File Buffer        |
          +--------------------+-------------------+
     +4   |   End Of Line      |    End Of File    |
          +--------------------+-------------------+
     +6   |     Text File      |    File State     |
          +--------------------+-------------------+
     +8   |             Record Length                |
          +--------------------+-------------------+
     +10  |    File Is Open    |  File Is Blocked  |
          +--------------------+-------------------+
     +12  |    Unit Number on which the File resides |
          +--------------------+-------------------+
     +14  | Length of Volume Name | Volume Name (7 bytes)...|
          +--------------------+-------------------+
     +22  |             Maximum Block                |
          +----------------------------------------+
     +24  |              Next Block                  |
          +----------------------------------------+
     +26  |             Repeat Count                 |
          +--------------------+-------------------+
     +28  | File Has Been Modified |     Unused      |
          +--------------------+-------------------+
     +30  |              First Block                 |
          +----------------------------------------+
     +32  |              Next Block                  |
          +---------+------------------------------+
     +34  | File Kind |            Unused            |
          +---------+----------+-------------------+
     +36  |Length Byte of Filename | Filename (15 bytes).....|
          +--------------------+-------------------+
     +52  |  Number of Bytes in the Last Block of the File |
          +---------+-----------+------------------+
     +54  | Month (4)| Day (5 bits)   | Year (7 bits)   |
          +---------+-----------+------------------+
     +56  |      Unused        | File has Soft Buffer |
          +--------------------+-------------------+
     +58  |              Maximum Byte                |
          +----------------------------------------+
     +60  |               Next Byte                  |
          +--------------------+-------------------+
     +62  |       Unused       | Buffer has been Changed |
          +--------------------+-------------------+
+64..571  | 512 byte buffer if the file has a 'soft buffer' |
          +--------------------+-------------------+
     +572 | 'window' big enough for one record of the file |
          +--------------------+-------------------+
```

2.6 Device Directory

A directory resides on a blocked device. The device directory contains information about the volume and the files that reside on that volume. A complete directory is an array of 73 directory entries, the first entry being the header record which describes the specific volume. The other 72 entries are for the files that reside on the device. The elements in a directory entry are described here:

FIRST BLOCK	is a word quantity which is the number of the first avaliable block on this device. This entry is normall zero (0).
NEXT BLOCK	is a word quantity which is the number of the next available block after this entry. For the volume header entry, this is normally 6.
FILE KIND	is a four-bit quantity which is the kind of file that this entry describes. The next two Subsections describe the different layouts of a directory entry depending on the file kind field. The values of file kind that are of interest are:

0	a directory header entry.
2	a code file.
3	a text file.
5	a data file.
8	is also a directory header entry.

the file kind entry is followed by 12 bits of unused space to fill up the word.

2.6.1 Directory Entry for a Header Record

If the FILE KIND field in the directory entry indicates that this entry is a directory header record, the following fields are valid:

VOLUME NAME	is an 8-byte field consisting of a length byte followed by seven characters of the volume name.
LAST BLOCK	a word quantity which is the number of the last available block on this volume.
NUMBER OF FILES	a word quantity which is the number of files on this volume.
LOAD TIME	a word quantity which is not used - it is set to zero.
LAST BOOT	is a word quantity which contains the most recent setting of the date. This word is in fact a date record.
MEMORY FLIPPED	a **Boolean** quantity only used by the system.
DISK FLIPPED	a **Boolean** quantity only used by the system.

There are two unused bytes at the end of the directory header entry.

2.6.2 Directory Entry for a File Entry

If the FILE KIND field in the directory entry indicates that this entry is any sort of file, the layout of the entry is as follows:

FILE NAME	is a 16-byte field containing the file name. The first byte contains the length of the field - the remaining 15 bytes are the characters of the file name.
LAST BYTE	is a word quantity which is the number of bytes in the last block of the file.
LAST MODIFICATION DATE	is a word quantity containing a date record representing the last time that this file was changed.

The diagram below illustrates the layout of a single directory entry. The first section is common to all kinds of directory entries. Then the entries on the left hand side correspond to a directory header entry and those on the right hand side correspond to a file entry.

```
                    +------------------------------+
   Byte --> +0      |          FIRST BLOCK         |
                    +------------------------------+
            +2      |          NEXT BLOCK          |
                    +-----------+------------------+
            +4      | FILE KIND |      UNUSED      |
                    +-----------+------------------+
      +---------------------------+---------------------------+
  +6  |    DISK VOLUME NAME       |         FILE NAME         |  +6
      +---------------------------+                           |
 +14  |       LAST BLOCK          |                           |
      +---------------------------+                           |
 +16  |     NUMBER OF FILES       |                           |
      +---------------------------+                           |
 +18  |       LAST ACCESS         |                           |
      +---------------------------+                           |
 +20  |        LAST BOOT          |                           |
      +------------+--------------+---------------------------+
 +22  | MEM FLIPPED| DISK FLIPPED |         LAST BYTE         |  +22
      +------------+--------------+---------------------------+
 +24  |         UNUSED            |        LAST ACCESS        |  +24
      +---------------------------+---------------------------+
```

Figure 2-3
Layout of a Directory Entry

2.7 The Device or Unit Table

The Device (or Unit) Table contains the maximum number of devices in the first word of the table. The remainder of the table consists of an entry for each particular unit. The overall layout of the unit table is as shown in the diagram below.

```
               +-----------------------------+
Byte ---> 0    | Maximum Number of Devices   |
               +-----------------------------+
         +2    |     Entry for Device 0      |
               +-----------------------------+
        +20    |     Entry for Device 1      |
               +-----------------------------+
        +38    |                             |
               | ..... and so on until ...   |
               +-----------------------------+
        +nn    |  Entry for Device MAXDEV    |
               +-----------------------------+
```

Figure 2-4
Overall Layout of the Device Table

Each entry in the unit table contains the following information:

VALID OPERATION BITS
a word quantity which contains a bit-map that with bits "on" to specify those operations that are valid for this device. The bits in the first word in each entry have the following meanings:

1	this unit can perform a UNITREAD operation.
2	this unit can perform a UNITWRITE operation.
4	this unit can perform a UNITCLEAR operation.

8	this unit can perform a UNITBUSY operation.
16	this unit can perform a UNITSTATUS operation.

ADDRESS OF DRIVER	is a long word pointer to the driver code for this device.
BLOCKED	a **Boolean** which when **true**, indicates that this is a blocked device.
MOUNTED	a **Boolean** which when **true**, indicates that this device is mounted (a driver is assigned to it).
DEVICE NAME	an eight-byte field which is the name of the device. The first byte is the length of the string; the remaining seven bytes are the actual name of the device.
DEVICE SIZE	is a word quantity which is the number of 512-byte blocks on this device. For an unblocked device, it is set to the maximum integer, 32767.

The layout of each entry in the device table is as shown below.

```
                +-----------------------------+
Offset  +0      |    Valid Operation Bits     |
                +-----------------------------+
        +2      | Pointer to Driver Routine   |
                +--------------+--------------+
        +6      |   BLOCKED    |   MOUNTED    |
                +--------------+--------------+
        +8      |    Device Name occupies     |
                |        eight bytes          |
                +-----------------------------+
        +16     |        Device Size          |
                +-----------------------------+
```

Figure 2-5
Individual Device Table Entries

2.8 Input Output Result Codes

The IORESULT field in the System Communication Area contains a result code every time some input-output process is completed. The table below describes the codes and their meanings.

0	Good result. The operation completed successfully.
1	Bad Block. Usually due to CRC error on disk read.
2	Either a bad unit number, or there is no driver implemented for this unit.
3	The requested input output function is not valid for this device. For example, block write to the keyboard. Also happens when attempting to open an already open file.
4	Nebulous Hardware Error.
5	Lost Device - a previously accessed device went offline.
6	Lost File - a previously accessed file has disappeared from the file directory.
7	Invalid File Name.
8	No room left on the device for the file.
9	this usually indicates something diastrous occurred while doing the input-output - the device is off-line, for example.
10	No File - the named file does not exist.
11	Duplicate File - attempt to rewrite a file that already exists.
12	File is Already Open - An attempt to open a file that is already open.

13	File Not Open - Attempt to operate on a closed file.
14	Bad Format - Non-numeric data read in an Integer or Real read operation.
15	Ring Buffer Overflow.
16	Write Protect - attempt to write to a write protected device.
17	Seek Error - Seek on a file that is not a text file or a blocked file. Also seek to a negative record number.
64	Device Error of unknown origin.

2.9 Memory Layout under MERLIN on the 68000

```
Top of Memory  +--------------------------------+
               |       MERLIN Kernel Code       |
               +--------------------------------+
               |             Shell              |
               +--------------------------------+
               |          Process 1             |
               |          Process 2             |
               |  .....   Process n             |
               |              |                 |
               |              V                 |
               |              ^                 |
               |              |                 |
               |   Permanently Resident Code    |
               +--------------------------------+
               |        Kernel Globals          |
               +--------------------------------+
               |        Kernel Stack            |
               +--------------------------------+
               |        Shell Globals           |
               +--------------------------------+
               |        Shell Stack             |
               +--------------------------------+
               | Process 1 Globals and Stack    |
               +--------------------------------+
               |            Stack               |
               |              |                 |
               |              V                 |
               |              ^                 |
               |              |                 |
               |            Heap                |
   $1000 --->  +--------------------------------+
               |                                |
               +--------------------------------+
    $180 --->  |    Pointer to SYSCOM Area      |
    $100 --->  +--------------------------------+
               |         Trap Vectors           |
      $0 --->  +--------------------------------+
```

Figure 2-6
Memory Layout in MERLIN

2.10 Register Usage in MERLIN

Registers A4 .. A7 are reserved for system use as follows:

A4	holds the address of the overlay jump table.
A5	holds the address of the user global data.
A6	holds the base address of the local stack frame. A6 is undefined for a procedure at the outermost (main) level.
A7	holds the current stack top address.

All other registers are CLOBBERED when system calls are made.

2.11 Environment of A Running Program

The diagram below shows the run-time environment pointed to by register A5.

```
                +------------------------------+
     (A5)+20    |  ARGC (argument count)       |
                +------------------------------+
     (A5)+16    | ARGV (point to Arguments)    |
                +------------------------------+
     (A5)+12    | Pointer to Standard Output   |
                +------------------------------+
     (A5)+8     | Pointer to Standard Input    |
                +------------------------------+
     (A5)+4     |       Return Address         |
                +------------------------------+
(A5) --------> |       Old Copy of A5         |
                +------------------------------+
```

Figure 2-7
Environment of a Running Program

Chapter 3

System Calls

This Chapter provides a blow-by-blow description of the system call interfaces. In all cases, parameters are described in the order in which they must be pushed onto the stack. The last thing pushed onto the stack, in all cases, is the return address. The discussions below cover the following topics:

- Unit input-output.
- File input-output.
- Memory Management.

3.1 Unit input-output

Unit input-output is at the lowest level of the system input-output facilities. Unit input-output references the physical devices in terms of physical blocks (on a disk). There are five system interfaces for unit input-output, namely UNITREAD, UNITWRITE, UNITBUSY, UNITCLEAR and UNITSTATUS. They are described in the subsections that follow.

3.1.1 UNITREAD and UNITWRITE - Direct Unit Data Transfer

UNITREAD and UNITWRITE are used to transfer information between a memory buffer and a specific unit. Parameters are:

unit number	a word quantity representing the physical unit number involved in the transfer.
buffer address	a long word pointer to the memory buffer.
byte count	a word quantity representing the number of bytes

	to be transferred.
block number	a word quantity representing the physical block number to be read or written. In the case of character devices such as the keyboard or printer, the block number is ignored.
mode	a word quantity which is driver dependent. For example, in the UCSD Pascal system, one of the functions of mode is to inhibit special treatment of space compression indicators in the byte stream as it flies through the driver.

3.1.2 UNITBUSY - Check if Unit is Busy

UNITBUSY can be called to determine if the unit is busy, that is, whether it is ready for data transfer. Parameters are:

unit number	a word quantity which is the number of the unit involved.

UNITBUSY returns a result on the stack top. The result is a Boolean quantity which is true if the unit is busy, false if not busy.

3.1.3 UNITCLEAR - Reset a Unit

UNITCLEAR is called to "reset" a unit to a known initialized state. Parameters are:

unit number	a word quantity representing the number of the unit to be cleared.

3.1.4 UNITSTATUS - Return Status of Unit

UNITSTATUS is a catch-all procedure which, in addition to returning the status of the specified unit, can also be used to change unit parameters. Parameters are:

unit number	a word quantity representing the phsyical unit number involved.
buffer address	a long word pointer to the buffer used for transferring information between UNITSTATUS and the caller.

control — a word quantity representing a control parameter whose meaning is agreed upon between UNITSTATUS and any of its callers.

3.2 File input-output

This Section describes those facilities that deal with files. In order to use the File input-output facilities, it is neccessary to allocate a File Information Block (FIB). See Chapter 2 for the details of an FIB. If Blocked input-output is being used, a buffer must also be allocated for the data transfer operations. The buffer must be big enough to hold the number of blocks to be transferred at any time.

3.2.1 FINIT - Initialize a File

FINIT sets up a File Information Block when the file is opened. The Open File function (FOPEN) usually calls upon FINIT to do this. User programs do not normally need to call FINIT. Parameters are:

Pointer to FIB — a long word pointer to a File Information Block.

bytes in a record — a word quantity. There are special meanings attached to this parameter if it is zero or negative. If positive, it represents the number of bytes per record in the file. If zero or negative, it has the following meanings:

0 — this file is an interactive file - it is talking to a device such as a terminal. An interactive file is to all intents and purposes the same as a text file. There are some minor differences in the way that end-of-line is handled.

-1 — this file is a UCSD Pascal compatible file. It is normally declared as just file; (an untyped file), as opposed to a file of some-type;. With this file organization, the user must provide the buffer. Block input-output can only be done to such a file type. See the Subsection on Block input-output later on.

-2	this file is an ISO Standard Pascal compatible file. That is, a file of text;.

3.2.2 FGET and FPUT - Transfer File Data

FGET and FPUT are considered together, since the calls are identical except for the data transfer direction. There is only one parameter:

<u>pointer to FIB</u> a long word pointer to a File Information Block.

3.2.3 FOPEN - Open File

FOPEN "opens" a file ready for data transfer. Parameters are:

<u>Pointer to Filename</u>
a long word pointer to a character string which represents the name of the file to be opened. The maximum number of characters in a file name is 24 at present. This is composed of a seven-character volume name enclosed between slash characters "/", followed by a 15-character file name.

<u>Pointer to FIB</u> a long word pointer to a File Information Block.

<u>New File Indicator</u>
a **Boolean** quantity which, when true, indicates that this is a new file, and when false, indicates that this is an existing file.

3.2.4 FCLOSE - Close File

FCLOSE closes a file and severs the relationship between a program and a file. It flushes out any buffers. FCLOSE also disposes of the file in a manner determined by the <u>mode</u> parameter described below. Parameters are:

<u>Pointer to FIB</u> a long word pointer to a File Information Block.

<u>Mode</u>	a word quantity indicating the disposition of the file after it is closed. The modes are:	
	0	<u>normal</u> - if the file is an <u>old</u> file - it existed prior to this program run, it is saved (retained) in the file system. If the file is a <u>new</u> file - created during this program run, it is deleted or purged from the file system.
	1	<u>lock</u> - makes a file permanent in the file system, regardless of any conditions mentioned in case (0) above.
	2	<u>purge</u> - purges or removes this file from the file system when the file is closed.

3.2.5 READCHAR - Read a Character from a File

READCHAR reads a single character from a file. READCHAR only applies to interactive (mode 0), or text (mode -2) files. Parameters are:

<u>Pointer to FIB</u> a long word pointer to a File Information Block.

READCHAR returns a single byte value on the top of the stack.

3.2.6 WRITECHAR - Write a Character to a File

WRITECHAR writes a character to a file. There is a field width specification which can cause space filling. WRITECHAR only applies to interactive (mode 0), or text (mode -2) files. Parameters are:

<u>Pointer to FIB</u>	a long word pointer to a File Information Block.
<u>Character</u>	to be written is a <u>byte</u>.
<u>Size</u>	a word quantity representing a field width. If <u>size</u> is greater than one, the character is preceded with size-1 spaces.

3.2.7 SEEK - Position to a Specific Record in a File

SEEK positions a file to the start of a specific record. It is intended for use in random file addressing situations. For a text file, it positions to the specified byte in the file. The position is absolute within the file, not relative to the previous position. Parameters are :

Pointer to FIB	a long word pointer to a File Information Block.
Record Number	a long word quantity representing the record to position to. Records are numbered from 0.

3.2.8 BLOCKIO - Block input-output

BLOCKIO (Block oriented input-output) is used to read or write whole blocks on a file. BLOCKIO only applies to untyped files - files created in mode -1. The blocks in question are physical disk blocks. In MERLIN's universe of discourse, blocks are 512 bytes. Parameters are:

Pointer to FIB	a long word pointer to a File Information Block.
Pointer to Buffer	a long word pointer to a buffer containing the data to be read or written.
Block Count	a word quantity representing the number of blocks to be transferred.
Block Number	a word quantity representing the block number at which to start the transfer. Blocks are numbered from zero.
Read or Write Indicator	a **Boolean** quantity indicating a Read when **true**, or a Write when **false**.

Block input-output returns a word quantity on the stack top. If the value is non-zero it is the number of blocks actually transferred. It is important to note that this value may not always be the same as the number of blocks requested - this happens when an end-of-file is encountered. If the value is zero, it indicates some form of error, in which case IORESULT should be read from the System Communication Area and checked for an error code.

3.3 Memory Management

This section describes those MERLIN system calls dealing with dynamic allocation and de-allocation of memory. Memory Allocation is done on a heap. The heap grows upward from the end of the user program. The user stack grows downward from the top of memory. When the two collide, there is mutual annihilation.

3.3.1 NEW - Allocate Storage

NEW allocates storage on the heap. Parameters are:

Pointer to Storage	a long word pointer which points to another long word pointer. The second pointer receives the start address of the allocated storage, in the event that there is enough storage to allocate. Note that NEW always returns a pointer that is aligned to a word boundary.
Byte Count	a word quantity representing the number of bytes to be allocated. Note that if an odd number of bytes are requested, NEW rounds up to an even (word) number and allocates that number of bytes.

3.3.2 DISPOSE - De-Allocate Storage

DISPOSE currently acts as a no-op. It does not actually dispose of de-allocate storage as in some Pascal implementations. DISPOSE does, however, return a NIL pointer to the caller. Parameters are:

Pointer to Storage	a long word pointer that itself points to another long word pointer. This second pointer is the address of the region of storage to be de-allocated.
Byte Count	a word quantity representing the number of bytes to be freed. It must be the same number as that given to the NEW call as described above.

3.3.3 MARK and RELEASE - Mark Heap and Release Heap

MARK and RELEASE are used in conjunction to de-allo previously allocated storage. They are identical in t parameter requirements:

<u>Pointer to Storage</u>
a long word pointer that itself points another long word pointer. This second poin is the start address of the storage region to marked or released.

MARK is used to "remember" the current position of the top heap. RELEASE subsequently uses the point that MARK returns to cut the heap top back the previously MARK'ed position.

3.3.4 MEMAVAIL - Determine Available Memory

MEMAVAIL returns, on the stack top, a long word quantity whic is the number of free bytes available on the heap.

<u>3.4 GETDIR - Read a Directory</u>

GETDIR reads a directory if one is available. Parameters are:

<u>Pointer to Volume Name</u>
is a long word pointer to a string which represents the name of a volume whose directory is to be read.

<u>Pointer to Directory</u>
a long word pointer to an area of memory large enough to receive an entire directory.

<u>Device Blocked Indicator</u>
is a long word pointer to a Boolean quantity which is set true if the device is blocked device.

<u>Device Number</u> is a long word pointer to a word quantity which is set to the device number.

Device is Valid Indicator
a long word pointer to a **Boolean** quantity which is set to true is the device named by the first parameter above is actually on the system. If this parameter is assigned the value false, none of the previous three parameters are defined.

The interpretation of the various parameters of GETDIR is as follows:

- If Device-is-Valid is **false**, the device named by the first parameter is not on-line. In this case, none of the other parameters are meaningful.

- If Device-is-Valid is **true**, The Device-Number parameter is assigned the number of the unit associated with that volume.

- The Device-Blocked parameter is set to **false** if the device is not a blocked device (such as the /printer). In this case, the Directory parameter is meaningless. If the Device-Blocked parameter is set to **true**, the device is a blocked device, in which case the Directory parameter contains the directory read in from that volume.

Chapter 4

Writing a Unit Driver

This Chapter discusses the basic concepts of writing a unit driver for MERLIN, then shows an example of such a driver written in 68000 assembler code.

4.1 Calling Conventions

Unit driver parameters are passed in registers, as follows:

D0.W	Unit number. This parameter is useful for validity checking where a given unit driver can have more than one logical device assigned to a single physical unit (such as a disk).
D1.L	Address of Buffer to or from which the data transfer is to be made.
D2.W	Number of Bytes of data to be transferred.
D3.W	Block Number at which the transfer is to start. This is only applicable to blocked devices.
D4.W	Command determines what operation (UnitRead, UnitBusy and so on), that the driver is to perform. This parameter is described in detail below. This parameter is the only valid parameter passed to unit-clear or unit-busy.
D5.W	Mode is device dependent and controls operations such as whether data compression characters are to be recognized.

The result of the operation (IORESULT) is returned in register D7.W.

4.1.1 Unit Driver Command Parameter

The Command passed in register D4.W describes what operation is to be performed. The command values are summarized here and described in greater detail below. When a given driver gets control, the caller has already verified (from the unit table) that this command is valid for this particular unit driver. The values of the command are:

0 Install the driver - perform any required initialization.

1 Read from the unit.

2 Write to the unit.

3 Clear the unit - reset it to its initial state.

4 Test if unit is busy.

5 Return status of unit.

6 Unmount the unit.

Install	When MERLIN installs a unit, either at boot time or when a unit is explicitly assigned, it is called with the install parameter. The unit can perform any initialization code neccessary to set up cyclic buffers, place interrupt vectors and so on.
Read and Write	Are self-explanatory.
Clear	Initializes the device - clear pending interrupts and such.
Busy	Check if the unit is ready for data transfer.
Status	Return the status of the unit. This operation is device dependent.
Unmount	Unmount the unit. This is called when the unit is re-assigned a new driver or is de-assigned. At this time the unit driver should perform any clean up or restoring of interrupt vectors that might be neccessary.

4.2 A Sample Unit Driver

The code below shows an entire unit driver with explanatory notes interspersed. The driver represents a model to be followed in broad outline rather than slavishly. Note the use of a table of self-relative addresses which the driver uses to jump to its various sections. A driver organized in this way can be located anywhere in memory and is independent of location.

```
        IDENT    CDURTDRI
;
        GLOBAL   UARTDRIV
;
;  UARTDRIV - The NEC PD7201 UART Unit Driver
;
;  Parameters:  D0.W - Unit Number
;               D1.L - Address of Buffer
;               D2.W - Count
;               D3.W - Block Number
;               D4.W - Command
;               D5.W - Access Mode
;
;                          Input Parameters:         Result values:
;    Command         Unit  Addr  Count Block Mode  IORESULT   Busy
;
;    0 - Install     D0.W                          D7.W
;    1 - Read        D0.W  D1.L  D2.W  D3.W  D5.W  D7.W
;    2 - Write       D0.W  D1.L  D2.W  D3.W  D5.W  D7.W
;    3 - Clear       D0.W                          D7.W
;    4 - Busy        D0.W                          D7.W       D0.B
;    5 - Status      D0.W  D1.L  D2.W*          D7.W
;    6 - Unmount     D0.W                          D7.W
;
```

To interpret the table above, the unit number for this driver is always passed in register D0.W. All commands always return an IORESULT in register D7.W. UNITBUSY, for example, is the only one that passes a result back in register D0.B. The UNITREAD, UNITWRITE and UNITSTATUS commands all expect a buffer address in register D1.L and a byte count in register D2.W.

In the case of the Status command, the value in register D2.W is a control parameter and not a count.

The next piece of code is the entry for a unit driver, illustrating how the various sections of the driver are called depending on the specific command.

```
;
;           Entry point for the UART Driver.
;
UARTDRIV
         CLR.W    D7                ;  IORESULT := 0.
         MOVE.L   D1,A0             ;  A0 := Data buffer address.
         LEA      URTTABL,A1        ;  A1 := Base address of offset table.
         LSL.W    #1,D4             ;  D4 := Command*2 for word count.
         MOVE.W   0(A1,D4.W),D4     ;  D4 := Offset from URTTABL.
         JMP      0(A1,D4.W)        ;  Go to appropriate driver.
;
URTTABL  DATA.W   URTINST-URTTABL    ;  Install driver.
         DATA.W   URTRD-URTTABL      ;  Read from UART.
         DATA.W   URTWR-URTTABL      ;  Write to UART.
         DATA.W   URTCLR-URTTABL     ;  Clear UART.
         DATA.W   URTBSY-URTTABL     ;  Test if Busy.
         DATA.W   URTST-URTTABL      ;  Return status.
         DATA.W   URTUNMT-URTTABL    ;  Unmount driver.
```

The next few code sections illustrate the entry points and give a broad view of the operations performed.

```
;
;     Constants to define the UART base addresses.
;
UARTA    EQU  $600000          ;  UART A data register.
UARTAC   EQU  $600002          ;  UART A command register.
;
;
URTINST                              ;  URTINST - Install the Driver.
         MOVE      #UARTAC,A0        ;  A0 := UART A control register.
         MOVE.B    #18,(A0)          ;  Select register 0.
         MOVE.B    #18,(A0)          ;  Reset the whole UART.
         MOVE.B    #2,(A0)           ;  Select register 2.
         ....  more code to
            .... initialize the UART
         RTS                         ;  Return to the caller.
;
;
URTUNMT                              ;  URTUNMT - Unmount the driver.
         RTS                         ;  Nothing to do in this driver.
```

```
;
;
URTRD                       ;  URTRD - Read character(s) from UART A.
UrdLoop  SUBQ.W   #1,D2             ;  Any more characters wanted ?
         BMI.S    UrdExit           ;  No - return to caller.
UrdBusy  MOVE.B   UARTAC.L,D0       ;  D0 := UART status register.
         ANDI.B   #1,D0             ;  Check if receiver full.
         BEQ.S    UrdBusy           ;  No - wait until it is.
         MOVE.B   UARTA.L(A0)+      ;  Yes - move character to buffer.
         BRA.S    UrdLoop           ;  Go for next character.
UrdExit  RTS                        ;  Finished - return to caller.
;
;
URTWR                       ;  URTWR - Write character(s) to UART A.
UwrLoop  SUBQ.W   #1,D2             ;  Any more characters to write ?
         BMI.S    UwrExit           ;  No - return to caller.
         .... remaining logic similar
            .... to URTRD except for
               .... direction of transfer
UwrExit  RTS                        ;  Finished - return to caller.
;
;
URTCLR                              ;  URTCLR - Clear the UART driver.
         MOVE.B   UARTA.L,D0        ;  Read character if present.
         RTS                        ;  Return to caller.
;
;
URTBSY                      ;  URTBSY - See if character available.
         MOVE.B   UARTAC.L,D0       ;  D0 := UART status register.
         ANDI.W   #1,D0             ;  Check if receiver full.
         SNE      D0                ;  Make condition code into ...
         NEG.B    D0                ;  ... a Pascal Boolean.
         RTS                        ;  Return to caller.
;
;
URTST                               ;  UART status - nothing to do.
         RTS                        ;  Return to caller.
;
         END      UARTDRIV          ;  End of the whole driver.
```

Chapter 5

Interface Definitions in Pascal

This chapter shows the Pascal type definitions, and the procedure interfaces, to MERLIN. The information given here is the Pascal representation of the narrative information in the preceding Chapters.

5.1 Basic Constant and Type Definitions

```
Const
  BLOCKSIZE   = 512;    { number of bytes in a disk block          }
  VIDLENGTH   = 7;      { number of characters in a volume name    }
  TIDLENGTH   = 15;     { number of characters in a file name      }
  MAXDIR      = 72;     { max number of directory entries/volume   }
  MAXDEV      = 20;     { max number of devices on the system      }
  MAXJTABLE   = 22;     { number of entries in system call table   }
  MAXUTABLE   = 10;     { number of entries in user call table     }
  MAXPROCESS  = 10;     { max number of processes allowed          }
  SYSCOMPLOC  = $0180;  { System Communication Area Pointer        }
  LOCODELOC   = $0108;  { Lowest memory location pointer           }
  HICODELOC   = $010C;  { Highest memory location pointer          }

                        { File disposition codes                   }
  FNORMAL     = 0;
  FLOCK       = 1;
  FPURGE      = 2;
  FTRUNC      = 3;

Type
  string80 = string[80];
  dirrange = 0 .. MAXDIR;
  vid = string[VIDLENGTH];
  tid = string[TIDLENGTH];
  filekind = (UNTYPEDFILE, XDSKFILE, CODEFILE, TEXTFILE, INFOFILE,
              DATAFILE, GRAFFILE, FOTOFILE, SECURDIR);
```

5.1.1 Layout of the Date Record

```
Type
  daterec = packed record
              year : 0 .. 100;  {  100 => temporary file  }
              day : 0 .. 31;
              month : 0 .. 12;  {  0 => date not meaningful  }
            end;
```

5.1.2 Layout of a Directory Entry

```
Type
  direntry =
        packed record
          firstblock : integer;
          nextblock : integer;
          status : boolean;
          filler : 0 .. 2047;
          case fkind : filekind of
            SECURDIR, UNTYPEDFILE:
              (dvid : vid;            { disk volume name           }
               deovblock: integer;    { last block of volume       }
               dnumfiles: integer;    { number of files            }
               dloadtime: integer;    { time of last access        }
               dlastboot: daterec);   { most recent date setting   }
               MemFlipped: Boolean;   { TRUE if flipped in memory  }
               DskFlipped: Boolean;   { TRUE if flipped on disk    }
             XDSKFILE, CODEFILE, TEXTFILE,
             INFOFILE, DATAFILE, GRAFFILE,
             FOTOFILE:
               (dtid: tid;            { title of file               }
                dlastbyte: 1 .. BLOCKSIZE; { bytes in last block    }
                daccess: daterec);    { last modification date      }
        end;

  directory = array[dirrange] of direntry;
  pdirectory = ^directory;

  devrange = 0 .. MAXDEV;

  byte = -128 .. 127;
```

```
bytes = array[0 .. 9999] of byte;
pbytes = ^bytes;
ppointer = ^pbytes;
string32 = string[32];
string64 = string[64];
pstring64 = ^string64;
str64rec = record s:string64; end;
pstr64rec: = ^str64rec;
stringtable = array[1 .. 100] of pstr64rec;
pstringtable = ^stringtable;
addrtable = array[0 .. MAXJTABLE] of pbytes;
paddrtable = ^addrtable;
uaddrtable = array[0 .. MAXUTABLE] of pbytes;
puaddrtable = ^uaddrtable;

memrec = record lodata: longint;
                hidata: longint;
                locode: longint;
                hicode: longint;
                btdev: integer;
         end;
```

5.1.3 File Interface Block Definition

```
type
 pfib = ^fib;
 fib = record fwindow: pbytes;
         FEOLN: Boolean;
         FEOF: Boolean;
         FTEXT: Boolean;
         fstate: (FTVALID, FIEMPTY, FIVALID, FTEMPTY);
         frecsize: integer;
         case FIsOpen:  Boolean of
          true: (FIsBlocked: Boolean;
                 funit: integer;
                 fvid: vid;
                 frepeatcount,
                 fnextblock,
                 fmaxblock: integer;
                 FModified: Boolean;
                 fheader: direntry;
                 case FSoftBuf: Boolean of
                  true: (fnextbyte, fmaxbyte: integer;
                         FBufChanged: Boolean;
                         fbuffer: array[0..511] of byte;
                         fuparrow: integer));
      end;
```

5.1.4 System Communication Area Definition

```
type
 ptext = ^text;

 syscomrec = record  sioresult: integer;
                     processno: integer;
                     freeheap: pbytes;
                     jtable: paddrtable;
                     sysout: ptext;
                     sysin: ptext;
                     sysdevtab: pdevtable;
                     pdirname: pstring64;
                     utable: puaddrtable;
                     today: daterec;
                     codejtaddr: longint;
                     nextprono: integer;
                     numpros: integer;
                     protable: pproctable;
                     pbootname: pstring64;
                     memmap: ^memrec;
                     bootdev: integer;
             end;
```

5.1.5 Layout of the Device Table

```
Type
  devrange = 0 ... MAXDEV;

  pdevtable = ^devtabrec;

  devtabrec = record  maxdevno: integer;
                   dt: array[devrange] of
                       record comnds: integer;
                           driver: pbytes;
                           Blocked: Boolean;
                           Mounted: Boolean;
                           devname: vid;
                           devsize: integer;
                       end;
              end;
```

5.1.6 Layout of the Process Table

```
Type
  pprocrec = ^procrec;

  procrec = record  d: array[0 .. 7] of longint;
                    a: array[0 .. 7] of longint;
                    no: integer;
            end;

  pproctable = ^proctable;

  proctable = array[0 .. MAXPROCESS] of procrec;
```

5.2 Procedure Interfaces in PASCAL

5.2.1 Unit Input Output

```
Procedure UNITREAD(unitno: Integer;
            buffer:pbytes;
            count: Integer;
            blockno: Integer;
            mode: Integer);

Procedure UNITWRITE(unitno: Integer;
            buffer:pbytes;
            count: Integer;
            blockno: Integer;
            mode: Integer);

Procedure UNITCLEAR(unitno: Integer);

Function UNITBUSY(unitno: Integer): Boolean;

Procedure UNITSTATUS(unitno: integer;
                     var buffer: pbytes;
                     control: integer);

Procedure UIOINIT;
```

5.2.2 File Input Output

```
Procedure  FINIT(f: pfib;  recbytes: integer);

procedure  FGET(f: pfib);

procedure  FPUT(f: pfib);

procedure  FOPEN(fpathname: pstring64;
                 f: pfib:
                 NewFlag: Boolean);

procedure  FCLOSE(f: pfib;  fmode: integer);

function  FREADCHAR(f: pfib): byte;

procedure  FWRITECHAR(f: pfib;  ch: byte;  fsize: integer);

procedure  FSEEK(f: pfib;  frecno: longint);

function  BLOCKIO(f: pfib;
                  fbuff: pbytes;
                  fblocks, fbock: integer;
                  ReadFlag: Boolean): integer;
```

CORVUS CONCEPT

Linker Librarian Reference Manual

LINKER and LIBRARY UTILITY

Reference Manual

First Edition

22nd December 1981

Silicon Valley Software Incorporated
10340 Phar Lap Drive
Cupertino
California 95014

This Linker and Library Utility Reference Manual was produced by:

Jeffrey Barth, R. Steven Glanville and Henry McGilton.

Silicon Valley Software Incorporated
Publication Number 810601-01

Table of Contents

Chapter 1 Introduction 1

1.1 Building an Executable Program 1
1.2 Overview and Layout of this Manual 2

Chapter 2 Linker 3

2.1 Linker Options 4
2.2 Linker Error Messages 5
2.3 Partial Linking 5

Chapter 3 Library Utility 7

Chapter 4 Object File Formats 9

4.1 Notation Used to Describe Object File Formats 9
4.2 Linker File Layout 9
4.3 Byte Level Description of Linker Blocks 10

4.3.1 80 - Module Name Block 11
4.3.2 81 - End Block 12
4.3.3 82 - Entry Point Block 13
4.3.4 83 - External Reference Block 14
4.3.5 84 - Starting Address Block 15
4.3.6 85 - Code Block 15
4.3.7 86 - 32-Bit Relocation Block 16
4.3.8 87 - Common Block Reference 17
4.3.9 88 - Common Block Definition 18
4.3.10 89 - Short External Reference Block 19
4.3.11 8A - FORTRAN Data Area Definition Block 20
4.3.12 8B - FORTRAN Data Area Initialization Block 21
4.3.13 8C - FORTRAN Data Area Reference Block 22
4.3.14 8E - Quick Load Executable Block 23
4.3.15 8F - Executable Block Definition 24
4.3.16 90 - Library Module Block 26
4.3.17 91 - Library Entry Block 27
4.3.18 92 - Unit Block 28
4.3.19 93 - FORTRAN Executable Data Area Reference Block 29
4.3.20 94 - FORTRAN Executable Data Area

Initialization Block 30
4.3.21 Text Block 31
4.3.22 EOF Mark 31

4.4 Executable Block Details 32

4.4.1 Layout of an Executable Block 32
4.4.2 Format of the Jump Table 34
4.4.3 Layout of a Segment Table 35
4.4.4 Layout of Descriptors 36

4.5 Loading a Segment 37
4.6 Running a Program 37

Chapter 1

Introduction

The Linker and Library utilities are a pair of complementary programs which aid in the process of generating executable programs under the MERLIN operating system.

The Linker links or binds relocatable object-code modules, and optional modules from libraries, to form a program which is executable.

The Library utility builds a library from relocatable object-code modules. Such a library can contain frequently used procedures (such as the mathematical functions of FORTRAN) which can be used in subsequent link processes.

1.1 Building an Executable Program

To get from the source text of a program to an executable object code file, the user must proceed as follows:

1. The source file is compiled or assembled. The result of compiling or assembling is a self-relocatable object-code file, along with listings and error diagnostics. This process continues until a "clean" compilation or assembly is obtained.
2. The relocatable object-code is linked, possibly including run-time support libraries, to generate executable code into a disk file.
3. The program can then be run (executed) on the machine simply by typing its filename.

The following chapters in this manual describe the Linker and Librarian object-code management system.

1.2 Overview and Layout of this Manual

Chapter 2 covers the Linker, its use, options and messages.

Chapter 3 describes the Library management utility and how to use it to build a library of relocatable object-code modules.

Chapter 4 is a detailed description of how object-code files are constructed, together with details of the various types of blocks that go to make an object-code file.

Chapter 2

Linker

The Linker is a utility which accepts files of relocatable object-code generated by the various compilers and assemblers, plus library files generated by the Library utility, and links or binds those into a form suitable for execution.

The Linker can also perform a partial link, where a collection of relocatable object-modules is bound into one file that can be used in future linking operations. This is described later on in this section.

As well as binding together relocatable modules from various language processors, the Linker can search libraries of commonly used functions, (such as the PASCAL run time environment), and link those modules that are referenced into the final loadable output file.

In order to link relocatable modules into an executable object-code file, the Linker needs the following pieces of information:

- The optional name of the listing file where the Linker messages and memory map information is to be listed. If no listing file name is given, no memory map information is generated.
- The name of the object-code file in which to write the final linked output.
- The name(s) of the file(s) from which the relocatable object-code is read.
- A list of one or more libraries which are to be used to satisfy external references within the object-code file.

A typical Linker run is shown below. Linker responses are in **bold face text**, and user input is underlined.

Example of Linker Usage

```
% linker
LINKER - MC68000 Object Code Linker
20-Jul-81
(C) 1981 Silicon Valley Software, Inc.

Listing File - /console
Output file[.OBJ] - myproglinked
Input file[.OBJ] - myprog
Input file[.OBJ] - paslib
Input file[.OBJ] -
     ..... Lots of Linker Messages .....
%
```

The Linker keeps prompting for more "Input files" until an empty line (carriage return) is entered. This enables the entry of a whole list of libraries as places from which to satisfy external references. The last one entered is usually the name of a run-time library (PASLIB in this example). A ".obj" suffix is added to all input filenames if it is omitted from the filename when entered.

If the Linker cannot find a specific input file, it displays a message to the effect:

```
*** Warning - Can't open input file ***
```

and repeats the prompt for an input file. The incorrect filename is simply ignored and the link can be completed with no adverse consequences.

2.1 Linker Options

Linker options are supplied on the command line when the Linker is called up. Linker options are introduced by a "+" sign, a "-" sign, followed by a letter, or a "?". The options are as follows:

? Display status information.

q The -q option disallows quick-load format for the executable object-code file, and forces overlay format. The +q option (the default) allows quick-load format.

u The +u option lists unreferenced entry points. The default is -u.

m The +m option prints the memory map in the order in which modules are linked. The default is -m.

a The +a option prints the memory map in alphabetical order. The default is +a.

s The +s option prints symbols that start with the "%" sign. Such symbols are used for compiler generated symbols. The default is -s or do not print "%" symbols.

2.2 Linker Error Messages

The Linker can display various error messages in the course of its operation. The error messages are self-explanatory. There are three grades of error messages, with different outcomes:

Warnings	are correctable errors. The error can be corrected and the link proceeds. For example, misspelling a filename will result in a message to the effect that the file cannot be opened, at which point the filename can be retyped.
Errors	are correctable in that the user can proceed with the link process, but the generated object-code file is not created properly.
Fatal errors	are those from which the Linker cannot correct or recover. In those cases the linker returns to the shell.

2.3 Partial Linking

As mentioned above, the Linker can perform a partial link, where the final output is not neccessarily executable, but a collection of separate relocatable object-code files can be combined into one file. The resultant file can then be used as an input file in subsequent link operations. The output of a partial link can have unsatisfied external references.

If, for any reason, the linked object file has not had all its external references satisfied, the linker displays a message to the effect:

```
    The output is not executable
```

This message appears when external references are not satisfied. It may mean that a program was missing some subroutines from a library (maybe the user forgot to include the library in the link process), or it also can appear when doing a partial link, in which case the message is to be ignored, since the full link will be done at a later date.

Chapter 3

Library Utility

The Librarian binds compiled or assembled relocatable object-code modules into a collection called a library. The purpose of a library is to provide a repository for commonly used object modules that have to be present when linking (see the Linker description), such that the common modules end up bound together into the final executable code module.

The library utility typically wants the following pieces of information form the user:

- The name of the file which is to receive the listing (results and log) of the library process.
- The name of the file which is to contain the generated library when the library generation process is complete.
- The name(s) of file(s) (with the .obj) suffix, which contain the constituent parts of the library to be generated.

A typical Librarian session appears below. Note that Librarian responses are in bold face text and user inputs are underlined.

```
% library
LIBRARY - MC68000 Library Utility
20-Jul-81
(C) 1981 Silicon Valley Software, Inc.

Listing file - /console
Output File[.OBJ] - bodleian
Input file[.OBJ] - bookshelf
Input file[.OBJ] - stacks
Input file[.OBJ] -
 .....  Lots of interesting Librarian messages .....
%
```

If the Librarian cannot find the specified input file it issues a message to the effect:

```
     The file 'whatever.obj' can't be opened
```

Chapter 4

Object File Formats

This chapter describes the layout of the object-code files that the Linker and Librarian can process. The various code blocks are described in sufficient detail that a compiler writer can generate object-code that is acceptable to the Linker and Librarian.

4.1 Notation Used to Describe Object File Formats

The symbol "::=" is read as "defined to be". Where a whole list of objects appear to the right of a "pile" of "::=" signs, it implies a choice of any of the objects.

Objects enclosed in "angle brackets", "<" and ">" are syntactic objects which are defined in terms of other objects.

An object followed by an asterisk sign, "*", can be repeated "zero to many times" (the list of objects can be empty).

An object followed by a plus sign, "+", can be repeated "one to many times" (there must be at least one of that object).

4.2 Linker File Layout

This section is a description of the Linker File at the "top level".

```
<Link File>     ::= <Module File>
                ::=   <Library File>
                ::=     <Unit File>
                ::=       <Execute File>

<Module File>   ::= <Module>* EOF mark

<Library File>  ::= <Library Module Block>+ <Library Entry Block>+
                    <Module>+  <Text Block>*  EOF Mark

<Unit File>     ::= <Unit Block> <Module>+ <Text Block> EOF Mark

<Execute File>  ::= <Executable Block> <Module>*
                ::=   <Quick Load Block>

<Module>        ::= <Module Name Block> <Other Block>+ <End Block>

<Other Block>   ::=  Entry Block
                ::=  External Block
                ::=  Start Block
                ::=  Code Block
                ::=  Relocation Block
                ::=  Common Relocation Block
                ::=  Common Definition Block
                ::=  Short External Block
                ::=  Data Initialization Block
                ::=  FORTRAN data area definition block
                ::=  FORTRAN data area Initialization Block
                ::=  FORTRAN Data Area Reference Block
                ::=  FORTRAN Executable Data Area Initialization Block
                ::=  FORTRAN Executable Data Area Reference Block
```

4.3 Byte Level Description of Linker Blocks

All Linker and Librarian object-code blocks start with a single "identifier byte". This block identifier takes values from 80 (base 16) upwards. The choice of values greater than 80 (base 16) is an attempt to minimise the probability that a regular ASCII text file is mistaken for the start of an object-code block.

4.3.1 80 - Module Name Block

```
               +--------+--------+--------+--------+
byte -->     0 |   80   |      size (3 bytes)      |
               +--------+--------+--------+--------+
             4 |           module name             |
               |            (8 bytes)              |
               +--------+--------+--------+--------+
            12 |          segment name             |
               |            (8 bytes)              |
               +--------+--------+--------+--------+
            20 |          csize (4 bytes)          |
               +--------+--------+--------+--------+
            24 | comments (24 .. size-1 bytes) ... |
               +--------+--------+--------+--------+
```

80	Hexadecimal 80 indicates a Module Name Block.
size	Number of bytes in this block.
module name	Blank padded ASCII name of module.
segment name	ASCII name of segment in which this module will reside.
csize	Number of bytes in the code block for this module.
comments	Arbitrary information - ignored by the Linker.

4.3.2 81 - End Block

```
byte -->     0 |   81    |      size (3 bytes)          |
               +---------+--------+--------+---------+
             4 |            csize (4 bytes)           |
               +---------+--------+--------+---------+
```

81	Hexadecimal 81 indicates this is an End Block.
size	Number of bytes in this block - it is always 000008.
csize	Number of bytes in the code block for this module.

4.3.3 82 - Entry Point Block

```
               +--------+--------+--------+--------+
byte -->     0 |   82   |      size (3 bytes)      |
               +--------+--------+--------+--------+
             4 |            link name              |
             8 |            (8 bytes)              |
               +--------+--------+--------+--------+
            12 |            user name              |
               |            (8 bytes)              |
               +--------+--------+--------+--------+
            20 |          loc (4 bytes)            |
               +--------+--------+--------+--------+
            24 | comments (24 .. size-1 bytes) ... |
               +--------+--------+--------+--------+
```

82	Hexadecimal 82 indicates this is an Entry Point Block.
size	Number of bytes in this block.
link name	Blank padded ASCII Linker name of entry point.
user name	Blank Padded ASCII user name of entry point.
loc	Location of entry point relative to this module.
comments	Arbitrary information - ignored by the Linker.

4.3.4 83 - External Reference Block

```
                   +--------+--------+--------+--------+
byte -->        0  |   83   |     size (3 bytes)       |
                   +--------+--------+--------+--------+
                4  |            link name              |
                8  |            (8 bytes)              |
                   +--------+--------+--------+--------+
               12  |            user name              |
                   |            (8 bytes)              |
                   +--------+--------+--------+--------+
               20  |         ref 1 (4 bytes)           |
                   +--------+--------+--------+--------+
               24  |         ref 2 (4 bytes)           |
                   +--------+--------+--------+--------+
                   |              . . .                |
                   +--------+--------+--------+--------+
                   | each reference consumes 4 bytes   |
                   +--------+--------+--------+--------+
                   |              . . .                |
                   +--------+--------+--------+--------+
         16+4*n    |         ref n (4 bytes)           |
                   +--------+--------+--------+--------+
```

83	Hexadecimal 83 indicates this is an External Reference Block.
size	Number of bytes in this block.
link name	Blank padded ASCII Linker name of external reference.
user name	Blank padded ASCII user name of external reference.
ref 1	Location of first reference relative to this module.
ref 2	Location of second reference relative to this module.
. . .	Other references.
ref n	Location of last reference relative to this module.

4.3.5 84 - Starting Address Block

```
                   +--------+--------+--------+---------+
byte -->         0 |   84   |     size (3 bytes)        |
                   +--------+--------+--------+---------+
                 4 |           start (4 bytes)          |
                   +--------+--------+--------+---------+
                 8 |           gsize (4 bytes)          |
                   +--------+--------+--------+---------+
                12 | comments (12 .. size-1 bytes) ...  |
                   +--------+--------+--------+---------+
```

84	Hexadecimal 84 indicates this is a Starting Address Block.
size	Number of bytes in this block.
start	Starting address relative to this module.
gsize	Number of bytes in the global data area.
comments	Arbitrary information - ignored by the Linker.

4.3.6 85 - Code Block

```
                   +--------+--------+--------+---------+
byte -->         0 |   85   |     size (3 bytes)        |
                   +--------+--------+--------+---------+
                 4 |           addr (4 bytes)           |
                   +--------+--------+--------+---------+
                 8 | object-code (8..size-1 bytes) ...  |
                   +--------+--------+--------+---------+
```

85	Hexadecimal 85 indicates this is a Code Block.
size	Number of bytes in this block.
addr	Module-relative address of first code byte.
object-code	The object-code - always an even number of bytes.

4.3.7 86 - 32-Bit Relocation Block

```
                  +--------+--------+--------+--------+
byte -->       0  |   86   |      size (3 bytes)      |
                  +--------+--------+--------+--------+
               4  |          addr 1 (4 bytes)         |
                  +--------+--------+--------+--------+
              12  |          addr 2 (4 bytes)         |
                  +--------+--------+--------+--------+
                  |               . . .               |
                  +--------+--------+--------+--------+
              16  |    each addr consumes 4 bytes     |
                  +--------+--------+--------+--------+
                  |               . . .               |
                  +--------+--------+--------+--------+
          12+4*n  |          addr n (4 bytes)         |
                  +--------+--------+--------+--------+
```

86	Hexadecimal 86 indicates this is a 32-bit Relocation Block.
size	Number of bytes in this block.
addr 1	Location of first address to relocate.
addr 2	Location of second address to relocate.
. . .	Locations of other addresses to relocate.
addr n	Location of last address to relocate.

4.3.8 87 - Common Block Reference

```
              +--------+--------+--------+--------+
byte -->    0 |   87   |     size (3 bytes)       |
              +--------+--------+--------+--------+
            4 |           common name             |
              |            (8 bytes)              |
              +--------+--------+--------+--------+
           12 |          ref 1 (4 bytes)          |
              +--------+--------+--------+--------+
           16 |          ref 2 (4 bytes)          |
              +--------+--------+--------+--------+
           20 |               . . .               |
              +--------+--------+--------+--------+
              | each reference consumes 4 bytes   |
              +--------+--------+--------+--------+
              |               . . .               |
              +--------+--------+--------+--------+
       8+4*n |          ref n (4 bytes)          |
              +--------+--------+--------+--------+
```

87	Hexadecimal 87 indicates this is a Common Block Reference.
size	Number of bytes in this block.
common name	Blank padded ASCII common block name.
ref 1	Location of first reference relative to this module.
ref 2	Location of second reference relative to this module.
. . .	Other references relative to this module.
ref n	Location of last reference relative to this module.

4.3.9 88 - Common Block Definition

```
                +--------+--------+--------+--------+
byte -->      0 |   88   |      size (3 bytes)      |
                +--------+--------+--------+--------+
              4 |            common name            |
                |            (8 bytes)              |
                +--------+--------+--------+--------+
             12 |          dsize (4 bytes)          |
                +--------+--------+--------+--------+
             16 | comments (16 .. size-1 bytes) ... |
                +--------+--------+--------+--------+
```

88	Hexadecimal 88 indicates this is a Common Block Definition.
size	Number of bytes in this block.
common name	Blank padded ASCII common data area name.
dsize	Number of bytes in this common data area.
comments	Arbitrary information - ignored by the Linker.

4.3.10 89 - Short External Reference Block

```
              +--------+--------+--------+--------+
byte -->    0 |   89   |      size (3 bytes)      |
              +--------+--------+--------+--------+
            4 |             link name             |
              |             (8 bytes)             |
              +--------+--------+--------+--------+
           12 |             user name             |
              |             (8 bytes)             |
              +--------+--------+--------+--------+
           20 | ref 1 (2 bytes) | ref 2 (2 bytes) |
              +--------+--------+--------+--------+
       18+2*n |      . . .      | ref n (2 bytes) |
              +--------+--------+--------+--------+
```

89	Hexadecimal 89 indicates this is a Short External Reference Block.
size	Number of bytes in this block.
link name	Blank padded ASCII Linker name of external reference.
user name	Blank padded ASCII user name of external reference.
ref 1	Location of first reference relative to this module.
ref 2	Location of second reference relative to this module.
. . .	Locations of other references relative to this module.
ref n	Location of last reference relative to this module.

The Linker does not yet support the short external reference block. It is intended to provide for one-word offsets that are either filled in with call-relative, short-absolute calls, or possibly calls indexed by an A-register, probably A4. The Linker will support this type of block in the future, and compilers will have an option to control the kind of generated call.

4.3.11 8A - FORTRAN Data Area Definition Block

```
                  +--------+--------+--------+--------+
byte -->      0   |   8A   |      size (3 bytes)      |
                  +--------+--------+--------+--------+
              4   |          data area name           |
                  |            (8 bytes)              |
                  +--------+--------+--------+--------+
             12   |          dsize (4 bytes)          |
                  +--------+--------+--------+--------+
```

8A	Hexadecimal 8A indicates this is a FORTRAN Data Area Definition Block.
size	Number of bytes in this block.
data area name	Blank padded ASCII name of FORTRAN fixed data area.
dsize	Size of this data area.

4.3.12 8B - FORTRAN Data Area Initialization Block

```
                  +--------+--------+--------+--------+
byte -->        0 |   8B   |      size (3 bytes)      |
                  +--------+--------+--------+--------+
                4 |           data area name          |
                  |              (8 bytes)            |
                  +--------+--------+--------+--------+
               12 |           daddr (4 bytes)         |
                  +--------+--------+--------+--------+
               16 | data occupies bytes 16 .. size-1  |
                  | in the rest of the block |  00 *  |
                  +--------+--------+--------+--------+
```

8B	Hexadecimal 8B indicates this is a FORTRAN Data Area Initialization Block.
size	Number of bytes in this block.
data area name	Blank padded ASCII name of FORTRAN fixed data area.
daddr	Starting address for this data.
data	The initialization data.
00 *	If the size of the data block is odd, there is one byte of 00 added to make the block an even number of bytes in size.

4.3.13 8C - FORTRAN Data Area Reference Block

```
                   +--------+--------+--------+--------+
byte -->        0  |   8C   |     size (3 bytes)        |
                   +--------+--------+--------+--------+
                4  |          data area name           |
                   |             (8 bytes)             |
                   +--------+--------+--------+--------+
               12  |          ref 1 (4 bytes)          |
                   +--------+--------+--------+--------+
               16  |          ref 2 (4 bytes)          |
                   +--------+--------+--------+--------+
                   |               . . .               |
                   +--------+--------+--------+--------+
                   | each reference consumes 4 bytes   |
                   +--------+--------+--------+--------+
                   |               . . .               |
                   +--------+--------+--------+--------+
           8+4*n   |          ref n (4 bytes)          |
                   +--------+--------+--------+--------+
```

8C	Hexadecimal 8C indicates this is a FORTRAN Data Area Reference Block.
size	Number of bytes in this block.
data area name	Blank padded ASCII name of FORTRAN fixed data area.
ref 1	Location of first reference.
ref 2	Location of first reference.
. . .	Location of other references.
ref n	Location of last reference.

4.3.14 8E - Quick Load Executable Block

```
                 +--------+--------+--------+--------+
byte -->      0  |   8E   |      size (3 bytes)      |
                 +--------+--------+--------+--------+
              4  |      start location (4 bytes)     |
                 +--------+--------+--------+--------+
              8  |        data size (4 bytes)        |
                 +--------+--------+--------+--------+
             12  | code block bytes (12..size-1) ... |
                 +--------+--------+--------+--------+
```

8E	Hexadecimal 8E indicates this is a Quick-Load Executable Block.
size	Number of bytes in this block.
start location	Relative starting address of the code block.
data size	Total number of bytes in global common data areas.
code block	The absolute, self-relocatable code block for this program.

4.3.15 8F - Executable Block Definition

```
                  +--------+--------+--------+--------+
byte -->        0 |   8F   |      size (3 bytes)      |
                  +--------+--------+--------+--------+
                4 |  jump table address (4 bytes)     |
                  +--------+--------+--------+--------+
                8 |    jump table size (4 bytes)      |
                  +--------+--------+--------+--------+
               12 |      data size (4 bytes)          |
                  +--------+--------+--------+--------+
               16 |       num       |   00   |   00   |
                  +--------+--------+--------+--------+
               20 |   00   |   00   |   00   |   00   |
                  +--------+--------+--------+--------+
               24 |          size 1 (4 bytes)         |
                  +--------+--------+--------+--------+
               28 |          size 2 (4 bytes)         |
                  +--------+--------+--------+--------+
                  |              . . .                |
                  +--------+--------+--------+--------+
           24+n*4 |          size n (4 bytes)         |
                  +--------+--------+--------+--------+
           28+n*4 | jump table bytes (... size-1) ... |
                  +--------+--------+--------+--------+
```

8F	Hexadecimal 8F indicates this is an Executable Block Definition.
size	Number of bytes in this block.
jump table address	Absolute load address of jump table.
jump table size	Number of bytes in the jump table.
data size	Total number of bytes in global common data areas.
num	Number of FORTRAN Data Areas.
00 00 00 00 00 00	six bytes of zero filler.
size 1	Size of first FORTRAN Data Area.

size 2	Size of second FORTRAN Data Area.
. . .	Sizes of other FORTRAN Data Areas.
size n	Size of last FORTRAN Data Area.
jump table	The jump table itself, including the executable code for the loader. For a further description, see the section on "Executable Block Details".

4.3.16 90 - Library Module Block

```
                +--------+--------+--------+--------+
byte -->      0 |   90   |     size (3 bytes)       |
                +--------+--------+--------+--------+
              4 |            module name            |
                |             (8 bytes)             |
                +--------+--------+--------+--------+
             12 |          msize (4 bytes)          |
                +--------+--------+--------+--------+
             16 |          caddr (4 bytes)          |
                +--------+--------+--------+--------+
             20 |          taddr (4 bytes)          |
                +--------+--------+--------+--------+
             24 |          tsize (4 bytes)          |
                +--------+--------+--------+--------+
             28 |  module count   |    module 1     |
                +--------+--------+--------+--------+
             32 |    module 2     |      . . .      |
                +--------+--------+--------+--------+
                |   module n-1    |    module n     |
                +--------+--------+--------+--------+
```

90	Hexadecimal 90 indicates this is a Library Module Block.
size	Number of bytes in this block.
module name	Name of this module.
msize	Number of bytes of code in this module.
caddr	Disk address of module.
taddr	If non-zero, is the disk address of the text block. If zero, there is no text block.
tsize	Size of text block.
module count	Number of other modules that this module references.
module 1	Number of the first module referenced.
module 2	Number of the second module referenced.

. . .	Numbers of other modules referenced.
module n	Number of the last module referenced.

4.3.17 91 - Library Entry Block

```
                    +--------+--------+--------+--------+
byte -->         0  |   91   |       size (3 bytes)     |
                    +--------+--------+--------+--------+
                 4  |             link name             |
                    |             (8 bytes)             |
                    +--------+--------+--------+--------+
                12  |     module      | . . . . . . . . .
                    +--------+--------+--------+--------+
                14  |          address (4 bytes)        |
                    +--------+--------+--------+--------+
```

4.3.18 92 - Unit Block

```
                  +--------+--------+--------+--------+
byte -->        0 |   92   |     size (3 bytes)       |
                  +--------+--------+--------+--------+
                4 |            unit  name             |
                  |             (8 bytes)             |
                  +--------+--------+--------+--------+
               12 |          caddr (4 bytes)          |
                  +--------+--------+--------+--------+
               16 |          taddr (4 bytes)          |
                  +--------+--------+--------+--------+
               20 |          tsize (4 bytes)          |
                  +--------+--------+--------+--------+
               24 |          gsize (4 bytes)          |
                  +--------+--------+--------+--------+
```

92	Hexadecimal 92 indicates that this is a Unit Block.
size	Number of bytes in this block - always 00001C.
unit name	Name of this unit.
caddr	Disk address of module.
taddr	Disk address of text block.
tsize	Size of text block.
gsize	Number of bytes of globals in this unit.

4.3.19 93 - FORTRAN Executable Data Area Reference Block

```
                   +--------+--------+--------+--------+
byte -->     0     |   93   |      size (3 bytes)      |
                   +--------+--------+--------+--------+
             4     |   area number   | . . . . . . . . .
                   +--------+--------+--------+--------+
             6     |           ref 1 (4 bytes)         |
                   +--------+--------+--------+--------+
            10     |           ref 2 (4 bytes)         |
                   +--------+--------+--------+--------+
                   |                . . .              |
                   +--------+--------+--------+--------+
                   | each reference consumes 4 bytes   |
                   +--------+--------+--------+--------+
                   |                . . .              |
                   +--------+--------+--------+--------+
        2+4*n      |           ref n (4 bytes)         |
                   +--------+--------+--------+--------+
```

93	Hexadecimal 93 indicates this is a FORTRAN Executable Data Area Reference Block.
size	Number of bytes in this block.
area number	Data area number.
ref 1	Address of first reference.
ref 2	Address of second reference.
. . .	Addresses of other references.
ref n	Address of last reference.

4.3.20 94 - FORTRAN Executable Data Area Initialization Block

```
                +--------+--------+--------+--------+
byte -->      0 |   94   |      size (3 bytes)      |
                +--------+--------+--------+--------+
              4 | data area number| . . . . . . . . .
                +--------+--------+--------+--------+
              6 |           daddr (4 bytes)         |
                +--------+--------+--------+--------+
             10 | initialization data . . . . . .   |
                +--------+--------+--------+--------+
                | . . . . . . . . . . . . . . . . . |
                +--------+--------+--------+--------+
                | . . . . . . . . . . . . |   00    |
                +--------+--------+--------+--------+
```

94	Hexadecimal 94 indicates this is a FORTRAN Executable Data Area Initialization Block.
size	Number of bytes in this block.
data area number	Number of the FORTRAN Data Area.
daddr	Starting address for this data.
initialization data	The data to fill the block with.
00	If the size of the initialization data is an odd number of bytes, a filler of 00 is appended to make it an even number of bytes.

4.4 Executable Block Details

This section describes the layout of an executable block. It includes details of the jump table and segment tables.

4.4.1 Layout of an Executable Block

```
                  +--------+--------+--------+--------+
byte -->       0  |   8F   |       size (3 bytes)     |
                  +--------+--------+--------+--------+
               4  |   Jump Table Address (4 bytes)    |
                  +--------+--------+--------+--------+
               8  |    Jump Table Size (4 bytes)      |
                  +--------+--------+--------+--------+
              12  |        Data Size (4 bytes)        |
                  +--------+--------+--------+--------+
              16  |       Num       |   00   |   00   |
                  +--------+--------+--------+--------+
              20  |   00   |   00   |   00   |   00   |
                  +--------+--------+--------+--------+
              24  |         Size 1 (4 bytes)          |
                  +--------+--------+--------+--------+
              28  |         Size 2 (4 bytes)          |
                  +--------+--------+--------+--------+
                  |               . . .               |
                  +--------+--------+--------+--------+
          20+4*n  |         Size n (4 bytes)          |
                  +--------+--------+--------+--------+
          24+4*n  | Jump Table (... size-1 bytes) ... |
                  +--------+--------+--------+--------+
```

8F	Hexadecimal 8F indicates this is an Executable Block Definition.
size	Number of bytes in this block.
jump table address	Absolute load address of jump table.
jump table size	Number of bytes in the jump table.

data size	Total number of bytes in global common data areas.
num	Number of FORTRAN Data Areas.
00 00 00 00 00 00	six bytes of zero filler.
size 1	Size of first FORTRAN Data Area.
size 2	Size of second FORTRAN Data Area.
. . .	Sizes of other FORTRAN Data Areas.
size n	Size of last FORTRAN Data Area.
jump table	The jump table itself, including the executable code for the loader.

If any FORTRAN Executable Data Area Initialization Blocks are present, they must immediately follow the executable block.

4.4.2 Format of the Jump Table

```
               +--------+--------+--------+--------+
A4 --> $$TOP   |   Number of Segments (2 bytes)    |
               +--------+--------+--------+--------+
            +2 |   Main Segment Table (32 bytes)   |
               +--------+--------+--------+--------+
           +34 |     Segment Table #2 (32 bytes)   |
               |       . . . . . . . . . .         |
               |     Segment Table #n (32 bytes)   |
               +--------+--------+--------+--------+
       2+n*32  |     Dummy Table #n+1 (4 bytes)    |
               +--------+--------+--------+--------+
               |   $_START Descriptor (10 bytes)   |
               +--------+--------+--------+--------+
               |     Segment #1 P#2 Descriptor     |
               |       . . . . . . . . . .         |
               |     Segment #1 P#n Descriptor     |
               +--------+--------+--------+--------+
               |     Segment #2 P#1 Descriptor     |
               |       . . . . . . . . . .         | All segment
               |     Segment #2 P#n Descriptor     | descriptors
               +--------+--------+--------+--------+ are 10 bytes.
               |     Segment #3 P#1 Descriptor     |
               +--------+--------+--------+--------+
               |               . . .               |
               +--------+--------+--------+--------+
               | Seg. #m P#n Descriptor (10 bytes) |
               +--------+--------+--------+--------+
           -20 |   Address of REMOVE1 (4 bytes)    |
               +--------+--------+--------+--------+
           -16 |    Address of Buffer (4 bytes)    |
               +--------+--------+--------+--------+
           -12 |  Address of Code File (4 bytes)   |
               +--------+--------+--------+--------+
            -8 |   Active Segment List (4 bytes)   |
               +--------+--------+--------+--------+
            -4 |    Address of $$TOP (4 bytes)     |
               +--------+--------+--------+--------+
     $$LOADIT  |     Object-code neccessary to     |
               |     load and execute a segment.   |
               +--------+--------+--------+--------+
```

4.4.3 Layout of a Segment Table

A Segment Table consists of eight 32-bit values:

```
                  +--------+--------+--------+--------+
byte -->       0  |   Address of first descriptor     |
                  +--------+--------+--------+--------+
               4  |     File Address of Segment       |
                  +--------+--------+--------+--------+
               8  |      Size of code in bytes        |
                  +--------+--------+--------+--------+
              12  |    Actual Address in Memory       |
                  +--------+--------+--------+--------+
              16  |     Scratch Return Address        |
                  +--------+--------+--------+--------+
              20  |     Segment Reference Count       |
                  +--------+--------+--------+--------+
              24  |     Active Segment-list link      |
                  +--------+--------+--------+--------+
              28  |  . . .     Reserved        . . .  |
                  +--------+--------+--------+--------+
```

4.4.4 Layout of Descriptors

An entry-point-descriptor is in one of two states, depending whether its corresponding segement is in memory or not. The formats of a descriptor are:

```
When Segment not in memory:            When segment in memory:

+-------------+--------------+         +-------------+--------------+
|  Relative offset of this   |         |  Relative offset of this   |
+---                    -----+         +---                     ----+
|   entry in its segment.    |         |   entry in its segment.    |
+-------------+--------------+         +-------------+--------------+
|          JSR xxx.L         |         |          JMP xxx.L         |
+-------------+--------------+         +-------------+--------------+
|    Absolute address of     |         |    Absolute address of     |
+---                    -----+         +---                     ----+
|          $$LOADIT          |         |    procedure as loaded     |
+-------------+--------------+         +-------------+--------------+
```

4.5 Loading a Segment

A segment is loaded into memory when the first call to one of its procedures is executed. Such a call is always via a descriptor in the jump table.

The JSR to $$LOADIT executes the loader from its entry-point '$$LOADIT'. The loader is able to tell which segement to load by comparing the place from which it was called with the limits of the segment-table entries found in the first part of the jump table. The loader then performs the following actions:

1. The loader loads that segment.
2. Fixes up all the JSR's to JMP's, so that further calls upon that segment jump directly to the entry-point instead of calling the loader.
3. Saves the calling routine's return address in the segment entry.
4. Patches the return address on the stack to return through the anti-loader entry-point '$$REMOVE1'.
5. Jump to the procedure entry-point which caused this loader invocation in the first place.

Further calls to entry-points in the segment are thus only slowed by a single JMP instruction instead of a loader call. When the initial call to that segment eventually returns, it will pass through '$$REMOVE1', which removes that segment and reclaims the memory which that segment uses.

4.6 Running a Program

When a program is executed, the program called 'run' performs the following steps:

1. The file containing the executable program is opened,

2. It is checked to see if it is the correct format, for example, the first byte should be $8F_{16}$,

3. The jump table is loaded into the proper location in memory, and

4. A JSR to JT+Word(JT)*32+2 is executed.

The normal overlay procedure then takes control to overlay the main segment and begin execution at its starting address.

*   CORVUS SYSTEMS

*   *

*   CORVUS CONCEPT TECHNICAL NOTES

Subject: Writing a Corvus CONCEPT Driver

Rev Lvl: 01 08-25-82 L. Franklin
02 11-08-82 L. Franklin (Update interrupt vectors)

This Technical Note discusses the basic concepts of writing a driver for Corvus CONCEPT Operating System, then shows an example of such a driver written in 68000 assembler code.

6.1 Driver Calling Conventions

Driver parameters are passed in registers, as follows:

D0.W	Unit number This parameter is useful for validity checking when a given driver can have more than one logical device assigned to a single physical unit (such as a disk).
D1.L	Buffer pointer Pointer to buffer to/from which the data transfer is to be made.
D2.W	Length of data transfer Number of bytes of data to be transferred.
D3.W	Starting block number Block number at which the transfer is to start. This is only applicable to blocked devices.
D4.W	Command Determines what operation (UnitRead, UnitBusy and so on), that the driver is to perform. This parameter is described in detail below. This parameter is the only valid parameter passed to UnitClear or UnitBusy.
D5.W	Mode Device dependent and control operations such as whether data compression characters are to be recognized.

The result of the operation (IORESULT) is returned in register D7.W by the called driver.

6.2 Driver Command Parameter

The command parameter passed in register D4.W defines the operation to be performed. When the driver gets control, the caller has already verified (from the device table) that this command is valid for this particular driver. The driver must have a minimum of an RTS for each command. Drivers must not use the heap or stack for long-term storage. The values of the command parameter are.

0	UnitInstall -- Install the driver When the operating system installs a unit, either at boot time or when a unit is explicitly assigned, the driver is called with the install parameter. This section performs any initialization code necessary to set up cyclic buffers, place interrupt vectors and so on.
1	UnitRead -- Read from the unit Self-explanatory.
2	UnitWrite -- Write to the unit Self-explanatory.
3	UnitClear -- Clear the unit Reset the device to its initial state. Initialize the device, clear pending interrupts and such.
4	UnitBusy -- Test if unit is busy Check if the unit is ready for data transfer. Driver returns D0.B = 1 (TRUE) if data is ready for transfer, D0.B = 0 (FALSE) otherwise.
5	UnitStatus -- Return status information from unit This command is device dependent. Using the function code (D2.W), the driver can return device dependent information to the caller. The buffer address may be used as a pointer to a UnitStatus parameter block.
6	UnitUnmount -- Unmount the unit This command is used when the unit is deassigned. At this time the driver must perform any clean up or restoring of interrupt vectors that might be necessary.

6.3 Static RAM Information

Each I/O slot is assigned a 256 byte area in static RAM. The RAM designated for each slot may be used in any manner by the device in the slot. Additionally, a 512 byte static RAM buffer is available for very temporary operations. This buffer may only be used during a single call to the driver. The static RAM locations are:

		Static RAM for I/O slots
00900-009FF	CPsl1ram	static RAM for slot 1 device
00A00-00AFF	CPsl2ram	static RAM for slot 2 device
00B00-00BFF	CPsl3ram	static RAM for slot 3 device
00C00-00CFF	CPsl4ram	static RAM for slot 4 device
		Buffers and stack
00D00-00EFF	CPiobuf	I/O buffer (512 bytes)

6.4 PROM Default Interrupt Vectors

The Corvus CONCEPT workstation PROM contains default interrupt handlers. If a driver uses system interrupts, the interrupt vector used by the driver must be restored when the driver is unmounted. The PROM also contains a table of default interrupt vectors which must be used when restoring an interrupt vector during unmount. The PROM locations for the default interrupt vectors are:

		Default interrupt vectors
10070-10073	CPivec1	level 1 interrupt vector (SLOTS)
10074-10077	CPivec2	level 2 interrupt vector (DC1)
10078-1007B	CPivec3	level 3 interrupt vector (OMNINET)
1007C-1007F	CPivec4	level 4 interrupt vector (DC0)
10080-10083	CPivec5	level 5 interrupt vector (TIMER)
10084-10087	CPivec6	level 6 interrupt vector (KYBD)
10088-1008B	CPivec7	level 7 interrupt vector

6.5 Driver Example

The code below shows an entire driver with explanatory notes interspersed. This driver represents a model to be followed in broad outline rather than slavishly. Note the use of a table of self-relative addresses which the driver uses to jump to its various sections. A driver organized in this way can be located anywhere in memory and is position independent. This is a requirement of the Corvus CONCEPT Operating System.

```
          IDENT    DRVDTACOM
          GLOBAL   DRVDTACOM

;
; DRVDTACOM - The DataComm driver
;
; Parameters:      D0.W - Unit number
;                  D1.L - Address of buffer
;                  D2.W - Count
;                  D3.W - Block Number
;                  D4.W - Command
;                  D5.W - Access Mode
;
;                       Input Parameters:                   Result values:
;     Command           Unit   Addr   Count  Block  Mode    IORESULT   Busy
;
;    0 - Install        D0.W                                 D7.W
;    1 - Read           D0.W   D1.L   D2.W   D3.W   D5.W     D7.W
;    2 - Write          D0.W   D1.L   D2.W   D3.W   D5.W     D7.W
;    3 - Clear          D0.W                                 D7.W
;    4 - Busy           D0.W                                 D7.W      D0.B
;    5 - Status         D0.W   D1.L   D2.W                   D7.W
;    6 - Unmount        D0.W                                 D7.W
;
```

The unit number for this driver is always passed in register D0.W. All commands always return an IORESULT in register D7.W. UnitBusy is the only command that passes a result back in register D0.B. The UnitRead, UnitWrite and UnitStatus commands all expect a buffer address in register D1.L and a byte count in register D2.W.

In the case of the UnitStatus command, the value in register D2.W is a control parameter or a UnitStatus function code and not a count.

The driver must protect A4-A6 which is used by the operating system.

The next section is the entry for a driver, illustrating how the Driver Header Table is organized.

```
;
; Driver related equates
;
IOEioreq equ     3                 ; IORESULT - invalid I/O request
MaxCmd   equ     6                 ; Maximum valid command
;
; Some UART equates
;
DcmPort  equ     $30f21            ; DataComm 0 UART pointer
Uda      equ     0                 ; UART data port offset
Ust      equ     2                 ; UART status port offset
RdBit    equ     3                 ; Busy bit for input
WrBit    equ     4                 ; Busy bit for output

;
; Entry point for the DataComm driver
;
DRVDTACOM
;
; Driver Header Table
;
         bra.s   DcmReq             ; Go to start of driver execution
         data.b  0                  ; Device blocked
                                    ;    (0 = false, 1 = true)
         data.b  15                 ; Valid commands (1-31)
         data.b  82,07,04,00        ; Date (year, month, day, filler)
         data.b  dhmlen             ; Length of ID message
dhm      data.b  'DataComm driver'; ID message
dhmlen   equ     %-dhm              ; Value of ID message length
```

The Driver Header Table is used by the operating system when loading the driver. It must be placed at the driver entry point and in the order shown above.

Valid commands range from 1 to 31 and are the summation of valid command codes for the driver. Command codes are:

```
    1 - UnitRead            4 - UnitClear          16 - UnitStatus
    2 - UnitWrite           8 - UnitBusy
```

Date and ID message are used to help track different versions of the driver. The ID message and message length (ie, a string) may be up to 30 characters in length.

The next section illustrates how the various sections of the driver are called depending on the specific command.

```
DcmReq  moveq    #IOEioreq,d7        ; Set IORESULT to invalid cmd
        cmp.w    #MaxCmd,d4          ; Is command valid?
        bhi.s    DcmRtrn             ; No, just return
        clr.w    d7                  ; Clear IORESULT
        movem.l  d1-d6/a0-a6,-(sp)   ; Save registers
        move.l   d1,a0               ; A0 := Data buffer address
        move.l   #DcmPort,a1         ; A1 := UART pointer
        lea      DcmTABL,a2          ; A2 := Offset table base addr
        lsl.w    #1,d4               ; D4 := Command*2 (word count)
        move.w   0(a2,d4.w),d4       ; D4 := Offset from DcmTABL
        jsr      0(a2,d4.w)          ; Call appropriate subroutine
        movem.l  (sp)+,d1-d6/a0-a6   ; Restore registers
                                     ;
DcmRtrn rts                          ; Return to caller

                                     ;
DcmTABL data.w   DcmINST-DcmTABL     ; Install driver
        data.w   DcmRD-DcmTABL       ; Read from DataComm
        data.w   DcmWR-DcmTABL       ; Write to DataComm
        data.w   DcmCLR-DcmTABL      ; Clear DataComm
        data.w   DcmBSY-DcmTABL      ; Test if busy
        data.w   DcmST-DcmTABL       ; Return status
        data.w   DcmUNMT-DcmTABL     ; Unmount driver
```

The next few code sections illustrate the entry points and give a broad view of the operations performed.

```
;
; DcmINST - Install the driver
;
DcmINST                             ;
          .... code to initialize
             .... the device
          rts                       ; Nothing to do in this example

;
; DcmUNMT - Unmount the driver
;
DcmUNMT                             ;
          .... code to terminate
             .... the device
          rts                       ; Nothing to do in this example

;
; DcmST  - Device dependent status request
;
DcmST                               ;
          .... code for status
             .... request
          rts                       ; Nothing to do in this example

;
; DcmCLR - Clear the DataComm driver
;
DcmCLR                              ;
          .... code to clear
             .... device
          rts                       ; Nothing to do in this example

;
; DcmBSY - See if character available
;
; Returns: D0.B - Result
;
DcmBSY  moveq    #0,d0              ; Assume FALSE (no character ready)
        btst     #RdBit,Ust(a1)     ; Character to read?
        boff.s   DcmBSYr            ; No, return
        moveq    #1,d0              ; Set TRUE
DcmBSYr rts                         ; Return
```

```
;
; DcmRD - Read character(s) from DataComm
;
DcmRD                               ;
CrdLoop subq.w   #1,d2              ; More to read?
        bmi.s    CrdExit            ; No, return
                                    ;
CrdBusy btst     #RdBit,Ust(a1)     ; Is char in UART?
        boff.s   CrdBusy            ; No, try again
        move.b   Uda(a1),(a0)+      ; Move character to buffer
        bra.s    CrdLoop            ; Go for next character
                                    ;
CrdExit rts                         ; Return

;
; DcmWR - Write character(s) to DataComm
;
DcmWR                               ;
CwrLoop subq.w   #1,d2              ; More to write?
        bmi.s    CwrExit            ; No, return
        .... remaining logic similar
            .... to DcmRD except for
                .... direction of transfer
CwrExit rts                         ; Return
```

Subject: Executing a Program from Corvus CONCEPT Pascal

Rev Lvl: 01 09-15-82 L. Franklin

This Technical Note explains the use of the Corvus CONCEPT Pascal CALL function which is used to execute a program from within a Pascal program. Also, the HALT procedure is described which is used by a Pascal program to set the CALL function result for the calling program. Both CALL and HALT are internal to the SVS Pascal compiler.

Refer to Technical Note 11 for a summary of callable system programs and their associated parameters (arguments).

10.1 CALL Function Parameters

The general form of the CALL function is:

```
    result := CALL (fileID, Ifile, Ofile, pArgPtrs, NbrArgs);
```

where:

result - is the function result of the called program. The function result is one of the following values:

< 0	execution error
0	no error
1	insufficient code memory
2	code file read error
3	file not executable code
4	file is not linked
5	code file open error
6	too many processes (10 maximum)
7	insufficient data memory
8	terminated by user

fileID - is a string containing the program file name. If the volume name is not specified, the current volume is searched for the specified program file. If the file is not found in the current volume, the system volume is then searched. If the volume name is specified, only that volume is searched for the program file. A program file name with a "!" prefix indicates the program file is in the system volume. These are the same rules as requesting a program from the command line.

Ifile	- is the default input file FIB for the called program. INPUT may be used to specify the default input file of the current program.
Ofile	- is the default output file FIB for the called program. OUTPUT may be used to specify the default output file of the current program.
pArgPtrs	- is a pointer to an array of argument string pointers. The array contains "NbrArgs" entries.
NbrArgs	- is the number of arguments to be passed to the called program (ARGC).

If function key labels are in use when the CALL function is executed, the calling program is responsible for turning off and reinitializing the function key labels. If the called program does not require user input, no special function key label processing is required.

10.2 CALL Function Example with No Arguments

The following example calls a program (MEM) with no arguments:

```
procedure callpgm;
    type   str64 = string[64];
          pstr64 = ^str64;
          strtbl = array [1..100] of pstr64;
         pstrtbl = ^strtbl;
    var  result: integer; p: pstrtbl;
    begin
    p := NIL;
    result := call ('/CCSYS/MEM',input,output,p^,0);
    end;
```

This example outputs a simple memory map to the console.

10.3 CALL Function Example with One Argument

The following example calls a program (SystmMgr) with one argument:

```
procedure callpgm;
     type   str64 = string[64];
           pstr64 = ^str64;
           strtbl = array [1..100] of pstr64;
          pstrtbl = ^strtbl;
     var  result: integer; s1: str64; p1: pstr64; p: pstrtbl;
     begin
     p := @p1; p1 := @s1; s1 := 'SETDAT';
     result := call ('!CC.SYSMGR',input,output,p^,1);
     end;
```

This example outputs the current date to the console.

10.4 CALL Function Example with Two Arguments

The following example calls a program (WndowMgr) with two arguments:

```
procedure callpgm;
     type   str64 = string[64];
           pstr64 = ^str64;
           strtbl = array [1..100] of pstr64;
          pstrtbl = ^strtbl;
     var  result: integer;
          s1,s2: str64; p1,p2: pstr64; p: pstrtbl;
     begin
     p := @p1; p1 := @s1; p2 := @s2;
     s1 := 'CSDISP'; s2 := '/CCUTIL/CSH.ALTCHARSET';
     result := call ('CC.WNDMGR',input,output,p^,2);
     end;
```

This example loads the alternate display character set.

In general, the pstr64 values (p1,p2,..,pn) must be declared in order since they become the argument string pointer array (the compiler allocates these variables sequentially).

10.5 HALT Procedure Parameters

The HALT procedure sets the CALL function result for the calling program and terminates program processing. The general form of the HALT procedure is:

```
    HALT (ReturnCode);
```

where ReturnCode is the integer function result value passed to the calling program. A zero ReturnCode is used to indicate a successful completion. Positive ReturnCode values are used by the Operating System during program loading. Negative ReturnCode values are used to indicate execution errors.

If an execution error (negative ReturnCode) is set during an EXEC file function, the remaining EXEC commands are ignored. System development programs, such as the Pascal compiler and linker, set execution error codes if the program function is not successfully completed. Therefore, an EXEC file with several Pascal compilations and links is terminated at the first execution error, saving time by not processing invalid data.

CORVUS CONCEPT DRIVER EXAMPLES

NOTE

THE EXAMPLES IN THIS FIRST SECTION ARE ACTUAL ASSEMBLY LANGUAGE SUBROUTINES FOUND IN CCLIB, THE PASCAL SYSTEM LIBRARY.

TABLE OF CONTENTS

CCLIB Library Subroutines

Driver Examples

Print Current Window Program Listing

Concept Keyboard Translation Tables

```
                        ; File: cclib.bit.text
                        ; Date. 13-May-82

                        ;
                        ; Corvus CONCEPT bit manipulation functions
                        ;

                                GLOBAL BITFLIP,BITSET,BITCLEAR,BITTEST,SHIFTRT,SHIFTLT,MAKEBYTE

                        ;
                        ; Function BitFlip (data,bitnum. integer). integer,
                        ;
0000  205F              BITFLIP MOVE.L  (SP)+,A0        , A0 = return address
0002  4C9F  0003                MOVEM.W (SP)+,D0-D1     , D0 = bit nmbr, D1 = data word
0006  0141                      BCHG    D0,D1           ; flip the bit
0008  3E81                      MOVE.W  D1,(SP)         , place changed word on stack
000A  4ED0                      JMP     (A0)            ; return to Pascal

                        ;
                        ; Function BitSet (data,bitnum. integer): integer,
                        ;
000C  205F              BITSET  MOVE.L  (SP)+,A0        ; A0 = return address
000E  4C9F  0003                MOVEM.W (SP)+,D0-D1     ; D0 = bit nmbr, D1 = data word
0012  01C1                      BSET    D0,D1           ; set the bit
0014  3E81                      MOVE.W  D1,(SP)         ; place changed word on stack
0016  4ED0                      JMP     (A0)            ; return to Pascal

                        ;
                        ; Function BitClear (data,bitnum: integer): integer,
                        ;
0018                    BITCLEAR
0018  205F                      MOVE.L  (SP)+,A0        ; A0 = return address
001A  4C9F  0003                MOVEM.W (SP)+,D0-D1     ; D0 = bit nmbr, D1 = data word
001E  0181                      BCLR    D0,D1           ; clear the bit
0020  3E81                      MOVE.W  D1,(SP)         ; place changed word on stack
0022  4ED0                      JMP     (A0)            , return to Pascal

                        ;
                        ; Function BitTest (data,bitnum: integer): boolean,
                        ;
0024  205F              BITTEST MOVE.L  (SP)+,A0        ; A0 = return address
0026  4C9F  0003                MOVEM.W (SP)+,D0-D1     , D0 = bit nmbr, D1 = data word
002A  4257                      CLR.W   (SP)            , assume false = 0
002C  0101                      BTST    D0,D1           ; test the bit
002E  6704                      BOFF.S  BTX             ,
0030  1EBC  0001                MOVE.B  #1,(SP)         , bit is on. return true
0034  4ED0              BTX     JMP     (A0)            ; return to Pascal

```

```
                 ;
                 ; Function ShiftRt (data: integer): integer;
                 ;
0036  205F       SHIFTRT MOVE.L  (SP)+,A0        ; A0 = return address
0038  301F               MOVE.W  (SP)+,D0        ; D0 = word to be shifted
003A  E248               LSR.W   #1,D0           ; shift it right
003C  3E80               MOVE.W  D0,(SP)         ; push result on stack
003E  4ED0               JMP     (A0)            ; return to Pascal

                 ;
                 ; Function ShiftLt (data: integer): integer;
                 ;
0040  205F       SHIFTLT MOVE.L  (SP)+,A0        ; A0 = return address
0042  301F               MOVE.W  (SP)+,D0        ; D0 = word to be shifted
0044  E348               LSL.W   #1,D0           ; shift it left
0046  3E80               MOVE.W  D0,(SP)         ; push result on stack
0048  4ED0               JMP     (A0)            ; return to Pascal

                 ;
                 ; Function MakeByte (n: integer): byte;
                 ;
004A             MAKEBYTE
004A  205F               MOVE.L  (SP)+,A0
004C  301F               MOVE.W  (SP)+,D0        ; get n
004E  1E80               MOVE.B  D0,(SP)         ; return function value
0050  4ED0               JMP     (A0)            ; return to Pascal

                         END

*BITCLEAR   000018+ *BITSET     00000C+  BTX        000034+ *SHIFTLT    000040+
*BITFLIP    000000+ *BITTEST    000024+ *MAKEBYTE   00004A+ *SHIFTRT    000036+

0 errors.  78 lines.
```

```
; File: cclib.asm.text
; Date: 06-Oct-82

;
; Corvus CONCEPT operating system interface
;

        IDENT   CCLIBASM
        GLOBAL  OSactSlt,OSactSrv,OSaltSlt,OSaltSrv,OSsltType
        GLOBAL  OSmaxDev,OSdispDv,OSkybdDv,OStimDv
        GLOBAL  OSomniDv,OSdcm2Dv,OSdcmlDv,OSsltDv
        GLOBAL  OSextCRT,pOSuserID,pOScurWnd,pOSsysWnd,pOSdevNam,pOSdate
        GLOBAL  xGetDir,xPutDir

        include '/ccos/os.gbl.asm.text'
```

```
                    ;
                    ; File. os.gbl.asm.text
                    ; Date. 20-Aug-82
                    ;
                    ;
                    ; Corvus CONCEPT operating system data structure equates
                    ;

                    ;
                    ; Additional Corvus CONCEPT I/O result codes
                    ;
00000003            IOEioreq equ    3       ; Invalid I/O request
                                            ;
00000015            IOEnotrn equ    21      ; Transporter not ready
00000016            IOEtimot equ    22      ; Timed out waiting for Omninet event
00000017            IOEnobuf equ    23      ; Read without a valid write buffer
                                            ;
00000020            IOEwndfn equ    32      ; Invalid window function
00000021            IOEwndbe equ    33      ; Window create boundary
00000022            IOEwndcs equ    34      ; Invalid character set
00000023            IOEwnddc equ    35      ; Delete current window
00000024            IOEwndds equ    36      ; Delete system window
00000025            IOEwndiw equ    37      ; Inactive window
00000026            IOEwndwr equ    38      ; Invalid window record
00000027            IOEwndwn equ    39      ; Invalid system window number
                                            ;
00000028            IOEnodsp equ    40      ; Display driver not available
00000029            IOEnokyb equ    41      ; Keyboard driver not available
0000002A            IOEnotim equ    42      ; Timer driver not available
0000002B            IOEnoomn equ    43      ; OMNINET driver not available
0000002C            IOEnoprt equ    44      ; Printer driver not available
0000002D            IOEnfdrv equ    45      ; No floppy drive at slot
                                            ;
00000032            IOEtblid equ    50      ; Invalid table entry ID
00000033            IOEtblfl equ    51      ; Table full
00000034            IOEtbliu equ    52      ; Table entry in use
00000035            IOEkybte equ    53      ; Keyboard transmission error
00000036            IOEuiopm equ    54      ; Invalid unit I/O parameter
00000037            IOEprmln equ    55      ; Invalid parameter block length
00000038            IOEfnccd equ    56      ; Invalid function code
00000039            IOEclkmf equ    57      ; Clock (hardware) malfunction

                    ;
                    ; System Common Pointer
                    ;
00000180            pSysCom    equ  $0180   ;pointer to address of SYSCOM
00000184            SysKybdFlg equ  $0184   ;keyboard control flags
00000186            SysByteScn equ  $0186   ;display driver - bytes per scan line

                    ;
                    ; System Common Equates
                    ;
00000000            SCiorslt equ    0       ;word - I/O result
00000002            SCprocno equ    2       ;word - current process number
```

```
00000004        SCfreehp equ    4       ;lint - free heap pointer
00000008        SCjtable equ    8       ;lint - jump table pointer
0000000C        SCsysout equ    12      ;lint - default output file pointer
00000010        SCsysin  equ    16      ;lint - default input file pointer
00000014        SCdevtab equ    20      ;lint - device (unit) table pointer
00000018        SCdirnam equ    24      ;lint - directory name string pointer
0000001C        SCutable equ    28      ;lint - user table pointer
00000020        SCtoday  equ    32      ;word - system date
00000022        SCcodejt equ    34      ;lint - code jump table pointer
00000026        SCnxtpro equ    38      ;word - next process number
00000028        SCnumpro equ    40      ;word - number of processes
0000002A        SCprotbl equ    42      ;lint - process table pointer
0000002E        SCbootnm equ    46      ;lint - boot device name pointer
00000032        SCmemmap equ    50      ;lint - memory map pointer
00000036        SCbootdv equ    54      ;word - boot device number
                                        ;
                                        ; CONCEPT additions
                ;        equ    56      ;word - unused
                ;        equ    58      ;word - unused
0000003C        SCslttbl equ    60      ;lint - slot table pointer
00000040        SCrootw  equ    64      ;lint - root window record pointer
00000044        SCcurrw  equ    68      ;lint - current window record pointer
00000048        SCcurrk  equ    72      ;lint - current keyboard record pointer
0000004C        SCuserid equ    76      ;word - Constellation user ID
0000004E        SCvrsnbr equ    78      ;lint - current version number string pointer
00000052        SCvrsdat equ    82      ;lint - current version date string pointer
00000056        SCwndtbl equ    86      ;lint - window table pointer
0000005A        SCsusinh equ    90      ;word - suspend inhibit count
0000005C        SCsusreq equ    92      ;word - suspend request if non-zero
```

```
                    ;
                    ; System Vector Equates
                    ;
00000000            SVuwrite equ     0*4     ;unit write
00000004            SVuread  equ     1*4     ;unit read
00000008            SVuclear equ     2*4     ;unit clear
0000000C            SVubusy  equ     3*4     ;unit busy
00000010            SVput    equ     4*4     ;put
00000014            SVget    equ     5*4     ;get
00000018            SVinit   equ     6*4     ;init
0000001C            SVopen   equ     7*4     ;open
00000020            SVclose  equ     8*4     ;close
00000024            SVwrchar equ     9*4     ;writechar
00000028            SVrdchar equ    10*4     ;readchar
0000002C            SVblkio  equ    11*4     ;blockio
00000030            SVseek   equ    12*4     ;seek
00000034            SVnew    equ    13*4     ;new
00000038            SVdspose equ    14*4     ;dispose
0000003C            SVmark   equ    15*4     ;mark
00000040            SVrlease equ    16*4     ;release
00000044            SVmavail equ    17*4     ;memory available
00000048            SVgetdir equ    18*4     ;get directory
00000060            SVcrkpth equ    24*4     ;crack path name
00000064            SVustat  equ    25*4     ;unit status
0000007C            SVcli    equ    31*4     ;command line interpreter
00000080            SVgetvnm equ    32*4     ;get volume names
00000084            SVvaldir equ    33*4     ;check valid directory
00000088            SVflpdir equ    34*4     ;flip directory
0000008C            SVschdir equ    35*4     ;search directory
00000090            SVdelent equ    36*4     ;delete directory entry
00000094            SVputdir equ    37*4     ;write directory
00000098            SVuinstl equ    38*4     ;unit install

                    ;
                    ; Memory Map Equates
                    ;
00000000            MMlodta  equ    0        ;lint - low data pointer
00000004            MMhidta  equ    4        ;lint - high data pointer
00000008            MMlocod  equ    8        ;lint - low code pointer
0000000C            MMhicod  equ    12       ;lint - high code pointer
00000010            MMbtsw   equ    16       ;word - boot switches
00000012            MMbtdev  equ    18       ;word - boot device number
00000014            MMbtslt  equ    20       ;word - boot slot number
00000016            MMbtsrv  equ    22       ;word - boot server number
00000018            MMbtdrv  equ    24       ;word - boot drive number
0000001A            MMbtblk  equ    26       ;word - boot volume block number

```

```
                 ;
                 ; Unit Table Equates
                 ;
00000002         UTiodrv equ    2       ;lint - I/O driver pointer
00000006         UTblf   equ    6       ;bool - blocked device flag
00000007         UTmtd   equ    7       ;bool - mounted device flag
00000008         UTdid   equ    8       ;str7 - device ID
00000010         UTsiz   equ    16      ;lint - device size
00000014         UTslt   equ    20      ;byte - device slot
00000015         UTsrv   equ    21      ;byte - device server
00000016         UTdrv   equ    22      ;byte - disk drive nmbr
00000017         UTtyp   equ    23      ;byte - disk drive type
00000018         UTspt   equ    24      ;byte - sectors per track
00000019         UTtps   equ    25      ;byte - tracks per side
0000001A         UTro    equ    26      ;bool - device read only
                 ;       equ    27      ;byte - ... unused
0000001C         UTblk   equ    28      ;lint - disk base block
00000020         UTlen   equ    32      ;       entry length

                 ;
                 ; Slot Table Equates
                 ;
00000000         STbtslt  equ   0       ;boot slot number
00000002         STbtsrv  equ   2       ;boot server number
00000004         STacslt  equ   4       ;active slot number
00000006         STacsrv  equ   6       ;active server number
00000008         STalslt  equ   8       ;alternate slot number
0000000A         STalsrv  equ   10      ;alternate server number
0000000C         STinfo   equ   12      ;array [1..5] of ....
                                        ;
00000000         STnmbr   equ   0       ;   slot number (1-5)
00000001         STtype   equ   1       ;   device type (slottypes)
00000002         STndrv   equ   2       ;   number of drives
00000004         STinfoL  equ   4       ;   device info length
```

```
                 ;
                 ; Character Set Record Equates
                 ;
00000000         CStblloc equ    0       ;character set data pointer
00000004         CSlpch   equ    4       ;scanlines per character (assume wide)
00000006         CSbpch   equ    6       ;bits per character (vertical height)
00000008         CSfrstch equ    8       ;first character code - ascii
0000000A         CSlastch equ    10      ;last character code - ascii
0000000C         CSmask   equ    12      ;mask used in positioning cells
00000010         CSattr1  equ    16      ;attributes
                                         ;  bit 0 = 1 - vertical orientation
00000011         CSattr2  equ    17      ;currently unused

                 ;
                 ; Window Record Equates
                 ;
00000000         WRcharpt equ    0       ;character set pointer
00000004         WRhomept equ    4       ;home (upper left) pointer
00000008         WRcuradr equ    8       ;current location pointer
0000000C         WRhomeof equ    12      ;bit offset of home location
0000000E         WRbasex  equ    14      ;home x value, relative to root window
00000010         WRbasey  equ    16      ;home y value, relative to root window
00000012         WRlngthx equ    18      ;maximum x value, relative to window (bits)
00000014         WRlngthy equ    20      ;maximum y value, relative to window (bits)
00000016         WRcursx  equ    22      ;current x value (bits)
00000018         WRcursy  equ    24      ;current y value (bits)
0000001A         WRbitofs equ    26      ;bit offset of current address
0000001C         WRgrorgx equ    28      ;graphics - origin x (bits relative to home loc)
0000001E         WRgrorgy equ    30      ;graphics - origin y (bits relative to home loc)
00000020         WRattr1  equ    32      ;attributes
                                         ;
00000000         invrse   equ    0       ;   inverse video mode
00000001         undscr   equ    1       ;   underscore mode
00000002         insmod   equ    2       ;   insert mode
00000003         viddeflt equ    3       ;   0 = W on B,       1 = B on W
00000004         noautolf equ    4       ;   0 = auto LF w/CR, 1 = no auto LF
00000005         syswin   equ    5       ;   system defined window
00000006         active   equ    6       ;   active window
00000007         suspend  equ    7       ;   suspended window
                                         ;
00000021         WRattr2  equ    33      ;attributes
                                         ;
00000000         vert     equ    0       ;   1 = vertical,   0 = horizontal screen
00000001         graphic  equ    1       ;   1 = graphics,   0 = character mode
00000002         curson   equ    2       ;   1 = cursor on,  0 = cursor off
00000003         invcurs  equ    3       ;   1 = inverse,    0 = underline cursor
00000004         wrapon   equ    4       ;   1 = wrap,       0 = clip at eoln
00000005         noscroll equ    5       ;   1 = no scroll,  0 = scroll
00000006         clrsc    equ    6       ;   1 = paging mode
00000007         vidset   equ    7       ;   1 = inverse     0 = normal
                                         ;
00000022         WRstate  equ    34      ;used for decoding escape sequences
00000023         WRrcdlen equ    35      ;window description record length
                                         ;
```

```
     00000024           240* WRlength equ    36       ;actual window record length
                        241*
```

```
                          
                           ;
                           ; OSACTSLT - Get active slot function
                           ;
                           ; FUNCTION OSactSlt: integer;
                           ;
0000                       OSactSlt
0000  2278  0180                   move.l  pSysCom.w,a1            ;Get pointer to SysCom
0004  2269  003C                   move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0008  3F69  0004  0004             move.w  STacslt(a1),4(sp)       ;Get active slot from slot table
000E  4E75                         rts                             ;Return
                          
                           ;
                           ; OSACTSRV - Get active server function
                           ;
                           ; FUNCTION OSactSrv: integer;
                           ;
0010                       OSactSrv
0010  2278  0180                   move.l  pSysCom.w,a1            ;Get pointer to SysCom
0014  2269  003C                   move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0018  3F69  0006  0004             move.w  STacsrv(a1),4(sp)       ;Get active server from slot table
001E  4E75                         rts                             ;Return
                          
                           ;
                           ; OSALTSLT - Get alternate slot function
                           ;
                           ; FUNCTION OSaltSlt: integer;
                           ;
0020                       OSaltSlt
0020  2278  0180                   move.l  pSysCom.w,a1            ;Get pointer to SysCom
0024  2269  003C                   move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0028  3F69  0008  0004             move.w  STalslt(a1),4(sp)       ;Get alternate slot from slot table
002E  4E75                         rts                             ;Return
                          
                           ;
                           ; OSALTSRV - Get alternate server function
                           ;
                           ; FUNCTION OSaltSrv: integer;
                           ;
0030                       OSaltSrv
0030  2278  0180                   move.l  pSysCom.w,a1            ;Get pointer to SysCom
0034  2269  003C                   move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0038  3F69  000A  0004             move.w  STalsrv(a1),4(sp)       ;Get alternate server from slot table
003E  4E75                         rts                             ;Return
```

```
                    ;
                    ; OSSLTTYPE - Get device type for slot function
                    ;
                    ; FUNCTION OSsltType (slot: integer): slottype;
                    ;
0040                OSsltType
0040  205F                  move.l  (sp)+,a0                ;Save return address
0042  301F                  move.w  (sp)+,d0                ;Get slot number
0044  548F                  addq.l  #2,sp                   ;Remove function result from stack
0046  5340                  subq.w  #1,d0                   ;Compute offset into slot table
0048  6D1C                  blt.s   slttyp8                 ;Error return if slot not valid
004A  0C40  0005            cmpi.w  #5,d0                   ;*
004E  6C16                  bge.s   slttyp8                 ;Error return if slot not valid
0050  C0FC  0004            mulu    #STinfoL,d0             ;*
0054  0640  000C            addi.w  #STinfo,d0              ;*
0058  2278  0180            move.l  pSysCom.w,a1            ;Get pointer to SysCom
005C  2269  003C            move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0060  1F31  0001            move.b  STtype(a1,d0.w),-(sp)   ;Get slot type for slot
0064  6002                  bra.s   slttyp9                 ;Return
                                                            ;
0066  4227          slttyp8 clr.b   -(sp)                   ;Set slot type to no device
                                                            ;
0068  4ED0          slttyp9 jmp     (a0)                    ;Return

                    ;
                    ; OSEXTCRT - Check for external CRT function
                    ;
                    ; FUNCTION OSextCRT: boolean;
                    ;
006A                OSextCRT
006A  205F                  move.l  (sp)+,a0                ;Save return address
006C  548F                  addq.l  #2,sp                   ;Remove function result from stack
006E  2278  0180            move.l  pSysCom.w,a1            ;Get pointer to SysCom
0072  2269  0014            move.l  SCdevtab(a1),a1         ;Get pointer to device table
0076  3019                  move.w  (a1)+,d0                ;Get number of devices
0078  2449                  move.l  a1,a2                   ;Compute last device pointer
007A  C0FC  0020            mulu    #UTlen,d0               ;*
007E  D5C0                  adda.l  d0,a2                   ;*
0080  2269  0002            move.l  UTiodrv(a1),a1          ;Get driver pointers
0084  246A  0002            move.l  UTiodrv(a2),a2          ;*
0088  7001                  moveq   #1,d0                   ;Assume TRUE
008A  B5C9                  cmpa.l  a1,a2                   ;Driver [0] = driver [MAXDEV]?
008C  6700  0004            beq     excrtx                  ;Yes, return
0090  7000                  moveq   #0,d0                   ;Set FALSE
0092  1F00          excrtx  move.b  d0,-(sp)                ;Set function result
0094  4ED0                  jmp     (a0)                    ;Return
```

```
                    ;
                    ; OSmaxDev - Get maximum device number function
                    ;
                    ; FUNCTION OSmaxDev: integer;
                    ;
0096                OSmaxDev
0096  2278  0180            move.l  pSysCom.w,a1          ;Get pointer to SysCom
009A  2269  0014            move.l  SCdevtab(a1),a1       ;Get pointer to device table
009E  3F51  0004            move.w  (a1),4(sp)            ;Get number of devices
00A2  4E75                  rts                           ;Return

                    ;
                    ; OSdispDv - Get DISPLAY driver device number function
                    ;
                    ; FUNCTION OSdispDv: integer;
                    ;
00A4                OSdispDv
00A4  4267                  clr.w   -(sp)                 ;Get number of devices
00A6  61EE                  bsr.s   OSmaxDev              ;*
00A8  301F                  move.w  (sp)+,d0              ;*
00AA  3F40  0004            move.w  d0,4(sp)              ;Set function result
00AE  4E75                  rts                           ;Return

                    ;
                    ; OSkybdDv - Get KYBD driver device number function
                    ;
                    ; FUNCTION OSkybdDv: integer;
                    ;
00B0                OSkybdDv
00B0  4267                  clr.w   -(sp)                 ;Get number of devices
00B2  61E2                  bsr.s   OSmaxDev              ;*
00B4  301F                  move.w  (sp)+,d0              ;*
00B6  5340                  subq    #1,d0                 ;Get device number
00B8  3F40  0004            move.w  d0,4(sp)              ;Set function result
00BC  4E75                  rts                           ;Return

                    ;
                    ; OStimDv - Get TIMER driver device number function
                    ;
                    ; FUNCTION OStimDv: integer;
                    ;
00BE                OStimDv
00BE  4267                  clr.w   -(sp)                 ;Get number of devices
00C0  61D4                  bsr.s   OSmaxDev              ;*
00C2  301F                  move.w  (sp)+,d0              ;*
00C4  5540                  subq    #2,d0                 ;Get device number
00C6  3F40  0004            move.w  d0,4(sp)              ;Set function result
00CA  4E75                  rts                           ;Return

                    ;
                    ; OSomniDv - Get OMNINET driver device number function
                    ;
                    ; FUNCTION OSomniDv: integer;
                    ;
```

```
00CC                 OSomniDv
00CC 4267                    clr.w   -(sp)             ;Get number of devices
00CE 61C6                    bsr.s   OSmaxDev          ;*
00D0 301F                    move.w  (sp)+,d0          ;*
00D2 5740                    subq    #3,d0             ;Get device number
00D4 3F40 0004               move.w  d0,4(sp)          ;Set function result
00D8 4E75                    rts                       ;Return

                     ;
                     ; OSdcm2Dv - Get DTACOM2 driver device number function
                     ;
                     ; FUNCTION OSdcm2Dv: integer;
                     ;
00DA                 OSdcm2Dv
00DA 4267                    clr.w   -(sp)             ;Get number of devices
00DC 61B8                    bsr.s   OSmaxDev          ;*
00DE 301F                    move.w  (sp)+,d0          ;*
00E0 5940                    subq    #4,d0             ;Get device number
00E2 3F40 0004               move.w  d0,4(sp)          ;Set function result
00E6 4E75                    rts                       ;Return

                     ;
                     ; OSdcm1Dv - Get DTACOM1 driver device number function
                     ;
                     ; FUNCTION OSdcm1Dv: integer;
                     ;
00E8                 OSdcm1Dv
00E8 4267                    clr.w   -(sp)             ;Get number of devices
00EA 61AA                    bsr.s   OSmaxDev          ;*
00EC 301F                    move.w  (sp)+,d0          ;*
00EE 5B40                    subq    #5,d0             ;Get device number
00F0 3F40 0004               move.w  d0,4(sp)          ;Set function result
00F4 4E75                    rts                       ;Return

                     ;
                     ; OSsltDv - Get SLOTIO driver device number function
                     ;
                     ; FUNCTION OSsltDv: integer;
                     ;
00F6                 OSsltDv
00F6 4267                    clr.w   -(sp)             ;Get number of devices
00F8 619C                    bsr.s   OSmaxDev          ;*
00FA 301F                    move.w  (sp)+,d0          ;*
00FC 5D40                    subq    #6,d0             ;Get device number
00FE 3F40 0004               move.w  d0,4(sp)          ;Set function result
0102 4E75                    rts                       ;Return
```

```
                              ;
                              ; pOSuserID - Get Constellation user ID pointer
                              ;
                              ; FUNCTION pOSuserID: pointer;
                              ;
0104                          pOSuserID
0104 2F78 0180 0004                   move.l  pSysCom.w,4(sp)        ;Get pointer to SysCom
010A 06AF 0000 004C                   addi.l  #SCuserID,4(sp)        ;Get pointer to user ID
0110 0004
0112 4E75                             rts                            ;Return

                              ;
                              ; pOScurWnd - Get current window record pointer
                              ;
                              ; FUNCTION pOScurWnd: pointer;
                              ;
0114                          pOScurWnd
0114 2078 0180                        move.l  pSysCom.w,a0           ;Get pointer to SysCom
0118 2F68 0044 0004                   move.l  SCcurrw(a0),4(sp)      ;Get current window pointer
011E 4E75                             rts                            ;Return

                              ;
                              ; pOSsysWnd - Get system window record pointer
                              ;
                              ; FUNCTION pOSsysWnd (wndnbr: integer): pointer;
                              ;
0120                          pOSsysWnd
0120 205F                             move.l  (sp)+,a0               ;Save return address
0122 301F                             move.w  (sp)+,d0               ;Get system window number
0124 2F08                             move.l  a0,-(sp)               ;Restore return address
0126 E548                             lsl.w   #2,d0                  ;Get index to window pointer
0128 2078 0180                        move.l  pSysCom.w,a0           ;Get pointer to SysCom
012C 2068 0056                        move.l  SCwndtbl(a0),a0        ;Get pointer to window table
0130 2F70 0000 0004                   move.l  0(a0,d0),4(sp)         ;Get window pointer
0136 4E75                             rts                            ;Return

                              ;
                              ; pOSdevNam - Get device name pointer
                              ;
                              ; FUNCTION pOSdevNam (unitnbr: integer): pointer;
                              ;
0138                          pOSdevNam
0138 205F                             move.l  (sp)+,a0               ;Save return address
013A 301F                             move.w  (sp)+,d0               ;Get unit number
013C C0FC 0020                        mulu    #UTlen,d0              ;Compute entry index
0140 2F08                             move.l  a0,-(sp)               ;Restore return address
0142 2078 0180                        move.l  pSysCom.w,a0           ;Get pointer to SysCom
0146 2068 0014                        move.l  SCdevtab(a0),a0        ;Get pointer to device table
014A D1FC 0000 0002                   adda.l  #2,a0                  ;Get pointer to device ID
0150 D1C0                             adda.l  d0,a0                  ;*
0152 D1FC 0000 0008                   adda.l  #UTdid,a0              ;*
0158 2F48 0004                        move.l  a0,4(sp)               ;Set function result
015C 4E75                             rts                            ;Return
```

```
                          ;
                          ; pOSdate - Get system date pointer
                          ;
                          ; FUNCTION pOSdate: pointer;
                          ;
015E                      pOSdate
015E  2F78  0180  0004            move.l  pSysCom.w,4(sp)        ;Get pointer to SysCom
0164  06AF  0000  0020            addi.l  #SCtoday,4(sp)         ;Get pointer to system date
016A  0004
016C  4E75                        rts                            ;Return
```

```
                    ;
                    ; JSVECT - Jump to routine in system vector
                    ;
                    ; Parameters: D0.W - offset in system vector
                    ;
016E  2078  0180    JSVECT  MOVE.L  pSysCom.W,A0      ; (A0) = syscom
0172  2068  0008            MOVE.L  SCjtable(A0),A0   ; (A0) = sysvect
0176  2070  0000            MOVE.L  0(A0,D0.W),A0     ; (A0) = desired routine
017A  4ED0                  JMP     (A0)              ; Go to it!

                    ;
                    ; JUVECT - Jump to routine in user vector
                    ;
                    ; Parameters: D0.W - offset in user vector
                    ;
017C  2078  0180    JUVECT  MOVE.L  pSysCom.W,A0      ; (A0) = syscom
0180  2068  001C            MOVE.L  SCutable(A0),A0   ; (A0) = uservect
0184  2070  0000            MOVE.L  0(A0,D0.W),A0     ; (A0) = desired routine
0188  4ED0                  JMP     (A0)              ; Go to it!

                    ;
                    ; XGETDIR - Read a directory
                    ;
                    ; procedure xgetdir (fvid: vid; var fdir: directory; var DevBlocked: Boolean;
                    ;                    var fdevno: integer; var DevValid: Boolean); external;
                    ;
018A  7048          XGETDIR MOVEQ   #SVgetdir,D0
018C  60E0                  BRA.S   JSVECT

                    ;
                    ; XPUTDIR - Write a directory
                    ;
                    ; procedure xputdir (var fdir: directory; fdevno: integer);
                    ;
018E  303C  0094    XPUTDIR MOVE.W  #SVputdir,D0
0192  60DA                  BRA.S   JSVECT

```

```
                    541*          END

ACTIVE    00000004  IOEWNDFN 00000020  SCBOOTDV 00000036  STNMBR   00000000  UTDID    00000008
CLRSC     00000006  IOEWNDIW 00000025  SCBOOTNM 0000002E  STTYPE   00000001  UTDRV    00000016
CSATTR1   00000010  IOEWNDWN 00000027  SCCODEJT 00000022  SUSPEND  00000007  UTIODRV  00000002
CSATTR2   00000011  IOEWNDWR 00000026  SCCURRK  00000048  SVBLKIO  0000002C  UTLEN    00000020
CSBPCH    00000006  JSVECT     00016E+ SCCURRW  00000044  SVCLI    0000007C  UTMTD    00000007
CSFRSTCH  00000008  JUVECT     00017C+ SCDEVTAB 00000014  SVCLOSE  00000020  UTRO     0000001A
CSLASTCH  0000000A  MMBTBLK  0000001A  SCDIRNAM 00000018  SVCRKPTH 00000060  UTSIZ    00000010
CSLPCH    00000004  MMBTDEV  00000012  SCFREEHP 00000004  SVDELENT 00000090  UTSLT    00000014
CSMASK    0000000C  MMBTDRV  00000018  SCIORSLT 00000000  SVDSPOSE 00000038  UTSPT    00000018
CSTBLLOC  00000008  MMBTSLT  00000014  SCJTABLE 00000008  SVFLPDIR 00000088  UTSRV    00000015
CURSON    00000002  MMBTSRV  00000016  SCMEMMAP 00000032  SVGET    00000014  UTTPS    00000019
EXCRTX      000092+ MMBTSW   00000010  SCNUMPRO 00000028  SVGETDIR 00000048  UTTYP    00000017
GRAPHIC   00000001  MMHICOD  0000000C  SCNXTPRO 00000026  SVGETVNM 00000080  VERT     00000000
INSMOD    00000002  MMHIDTA  00000004  SCPROCNO 00000002  SVINIT   00000018  VIDDEFLT 00000003
INVCURS   00000003  MMLOCOD  00000008  SCPROTBL 0000002A  SVMARK   0000003C  VIDSET   00000007
INVRSE    00000000  MMLODTA  00000000  SCROOTW  00000040  SVMAVAIL 00000044  WRAPON   00000004
IOECLKMF  00000039  NOAUTOLF 00000004  SCSLTTBL 0000003C  SVNEW    00000034  WRATTR1  00000020
IOEFNCCD  00000038  NOSCROLL 00000005  SCSUSINH 0000005A  SVOPEN   0000001C  WRATTR2  00000021
IOEIOREQ  00000003  *OSACTSLT  000000+ SCSUSREQ 0000005C  SVPUT    00000010  WRBASEX  0000000E
IOEKYBTE  00000035  *OSACTSRV  000010+ SCSYSIN  00000010  SVPUTDIR 00000094  WRBASEY  00000010
IOENFDRV  0000002D  *OSALTSLT  000020+ SCSYSOUT 0000000C  SVRDCHAR 00000028  WRBITOFS 0000001A
IOENOBUF  00000017  *OSALTSRV  000030+ SCTODAY  00000020  SVRLEASE 00000040  WRCHARPT 00000000
IOENODSF  00000028  *OSDCM1DV  0000E8+ SCUSERID 0000004C  SVSCHDIR 0000008C  WRCURADR 00000008
IOENOKYB  00000029  *OSDCM2DV  0000DA+ SCUTABLE 0000001C  SVSEEK   00000030  WRCURSX  00000016
IOENOOMN  0000002B  *OSDISPDV  0000A4+ SCVRSDAT 00000052  SVUBUSY  0000000C  WRCURSY  00000018
IOENOPRT  0000002C  *OSEXTCRT  00006A+ SCVRSNBR 0000004E  SVUCLEAR 00000008  WRGRORGX 0000001C
IOENOTIM  0000002A  *OSKYBDDV  0000B0+ SCWNDTBL 00000056  SVUINSTL 00000098  WRGRORGY 0000001E
IOENOTRN  00000015  *OSMAXDEV  000096+ SLTTYP8    000066+ SVUREAD  00000004  WRHOMEOF 0000000C
IOEPRMLN  00000037  *OSOMNIDV  0000CC+ SLTTYP9    000068+ SVUSTAT  00000049  WRHOMEPT 00000004
IOETBLFL  00000033  *OSSLTDV   0000F6+ STACSLT  00000004  SVUWRITE 00000000  WRLENGTH 00000024
IOETBLID  00000032  *OSSLTTYP  000040+ STACSRV  00000006  SVVALDIR 00000084  WRLNGTHX 00000012
IOETBLIU  00000034  *OSTIMDV   0000BE+ STALSLT  00000008  SVWRCHAR 00000024  WRLNGTHY 00000014
IOETIMOT  00000016  *POSCURWN  000114+ STALSRV  0000000A  SYSBYTES 00000186  WRRCDLEN 00000023
IOEUIOPM  00000036  *POSDATE   00015E+ STBTSLT  00000000  SYSKYBDF 00000184  WRSTATE  00000022
IOEWNDBE  00000021  *POSDEVNA  000138+ STBTSRV  00000002  SYSWIN   00000005  *XGETDIR   00018A+
IOEWNDCS  00000022  *POSSYSWN  000128+ STINFO   0000000C  UNDSCR   00000001  *XPUTDIR   00018E+
IOEWNDDC  00000023  *POSUSERI  000104+ STINFOL  00000004  UTBLF    00000006
IOEWNDDS  00000024  PSYSCOM  00000180  STNDRV   00000002  UTBLK    0000001C

0 errors.  542 lines.
```

NOTE

THE EXAMPLES IN THE FOLLOWING TWO SECTIONS ARE EXAMPLES ONLY. THEY DO NOT REFLECT THE CURRENT OPERATING SYSTEM.

```
;
; file : timer.drv.text
; date : 20-SEPTEMBER-1982  kb
;
, INCLUDE FILES USED :
;       timer.clk.text                         ,HAS CALANDER CLOCK CODE
;       /ccos/os.gbl.asm.text                  ;OS GLOBAL EQUATES
;
; 04-06-82  kb  Added version date before TIMERDRV - entry point
; 04-23-82  kb  Changed IORESULT definitions to use the global file definitions
; 04-23-82  kb  Removed volume name from timer.clk.text include
; 06-07-82  kb  Changed for new rev. 4 processor board changes, will find the
;                  correct address to use (either $30FE1 or $30F81)
;                  added storage loc to save correct address
; 07-07-82  kb  Added Header to driver
; 07-07-82  kb  Fixed error in equates for different rev board address equates
; 09-20-82  lf  Changed write clock to stop/start clock in order to zero
;                  seconds and tenths fields (in TIMER.CLK.TEXT)
;
;
; INCLUDE OS GLOBALS HERE
;
            LIST       ON
;
```

```
                    256* ;
                    ; EQUATES FOR ALL TIMER DRIVER SOFTWARE
                    ;
                    ; BIT NUMBER DEFINITIONS
                    ;
00000000            BITD0       EQU     0                       ;BIT 0
00000001            BITD1       EQU     1                       ;BIT 1
00000002            BITD2       EQU     2                       ;BIT 2
00000003            BITD3       EQU     3                       ;BIT 3
00000004            BITD4       EQU     4                       ;BIT 4
00000005            BITD5       EQU     5                       ;BIT 5
00000006            BITD6       EQU     6                       ;BIT 6
00000007            BITD7       EQU     7                       ;BIT 7
                    ;
                    ; TIMER INTERRUPT VECTOR ADDRESS
                    ,
00000074            VECTOR      EQU     $000074                 ;INTERRUPT VECTOR #5
                    ;
                    ; TIMER TABLE INDICES
                    ;
00000000            TFLAGS      EQU     0                       ,TIMER TABLE FLAGS
00000002            PTRUSRTN    EQU     2                       ;POINTER TO USER SERVICE ROUTINE
00000006            TCOUNT      EQU     6                       ,# OF 50 MS. TICKS BEFORE CALL
00000008            TDWNCNT     EQU     8                       ,WORKING DOWN COUNTER
0000000A            REGA4       EQU     TDWNCNT+2               ;REGISTER A4 SAVE AREA
0000000E            REGA5       EQU     REGA4+4                 ,REGISTER A5 SAVE AREA
                    ;
                    ; TIMER TABLE FLAGS BIT DEFINITIONS
                    ;
00000000            VALIDENT    EQU     BITD0                   ;VALID ENTRY FLAG
00000001            CONT1SHT    EQU     BITD1                   ;CONTINUOUS/1-SHOT MODE FLAG
00000002            SKIP1ST     EQU     BITD2                   ;SKIP FIRST CALL FLAG
00000003            ENBLDSBL    EQU     BITD3                   ;ENABLE/DISABLE FLAG
                    ;
                    ; BELL PARAMETER BLOCK INDICES
                    ;
00000000            FREQ        EQU     0                       ;FREQUENCY OF BELL
00000002            PATTERN     EQU     2                       ,PATTERN OF SPEAKER ON AND OFFS
00000004            DURATN      EQU     4                       ;DURATION OF BELL
                    ,
                    ; INTERNAL FLAG BIT DEFINITIONS
                    ;
00000000            SHUTOFF     EQU     BITD0                   ,SHUTOFF BELL FLAG
                    ;
                    ; VIA ADDRESSES
                    ;
00030F77            ACR         EQU     $30F77                  ;AUXILLARY CONTROL REGISTER
00030F7D            IER         EQU     $30F7D                  ;INTERRUPT ENABLE REGISTER
00030F7B            IFR         EQU     $30F7B                  ,INTERRUPT FLAGS REGISTER
00030F6D            T1LL        EQU     $30F6D                  ;TIMER 1 LATCH LOW
00030F6F            T1LH        EQU     $30F6F                  ;TIMER 1 LATCH HIGH
00030F69            T1CL        EQU     $30F69                  ;TIMER 1 COUNTER LOW - READ ONLY
00030F6B            T1CH        EQU     $30F6B                  ,TIMER 1 COUNTER HIGH
00030F71            T2LL        EQU     $30F71                  ,TIMER 2 LATCH LOW
00030F73            T2CH        EQU     $30F73                  ;TIMER 2 COUNTER HIGH
```

```
00030F75        SHIFTREG    EQU     $30F75              ;SHIFT REGISTER
                ;
                ; VIA REGISTER VALUES
                ;
00000040        ACRBYTE     EQU     $40                 ;ACR DATA - T1 FREE RUN DISABLE PB7
00000010        RUNT2       EQU     $10                 ;MASK TO COUNT DOWN T2
000000EF        STOPT2      EQU     $EF                 ;COMPLEMENTED RUNT2 TO STOP T2
0000007F        DISABL      EQU     $7F                 ;DISABLE ALL INTERRUPTS
000000C0        ENBLT1      EQU     $C0                 ;ENABLE IRQ FOR T1
000000FF        CLEAR       EQU     $FF                 ;CLEAR ALL IFR STAT BITS
00000020        T2INT       EQU     $20                 ;TIMER #2 INTERRUPT FLAG BIT
0000C350        TIME        EQU     50000               ;50,000 MICRO SECONDS
000000C3        TIMEH       EQU     TIME/256            ;HI ORDER BYTE OF TIME VALUE
00000050        TIMEL       EQU     TIME-(TIMEH*256)    ,LOW ORDER BYTE OF TIME VALUE
                ,
                ; CONTEXT SWITCHING DEFINITIONS
                ;
0000005C        SPNDFLG     EQU     SCsusreq            ;SUSPEND FLAG
0000005A        SPWAITC     EQU     SCsusinh            ;WAIT SUSPEND COUNTED SEMAPHORE
00000002        CURPROC     EQU     SCprocno            ;CURRENT PROCESS # INDEX
000000BB        PPTBL       EQU     $BB                 ;PTR TO PROCESS TABLE
000000BB        SCHEDPTR    EQU     $BB                 ;PTR TO ENTRY OF SCHEDULER
000000BB        SCHDA4      EQU     $BB                 ;REG. A4 VALUE FOR SCHEDULER
000000BB        SCHDA5      EQU     $BB                 ;REG. A5 VALUE FOR SCHEDULER
                ,
000000CC        PTLEN       EQU     $CC                 ;LENGTH OF PROCESS TABLE ENTRY
0000000F        NUMREGS     EQU     15                  ;NUMBER OF REGISTERS SSAVED IN PTBL
0000003C        PTPC        EQU     NUMREGS*4           ;PROCESS TABLE-PC FIELD
00000040        PTSR        EQU     PTPC+4              ;PROCESS TABLE-SR FIELD
                ;
00002700        SCHEDSR     EQU     $2700               ;SCHEDULER SR-NO INTERRUPTS
                ;
                ; IORESULT ERROR CODES
                ;
00000036        INVPRM      EQU      IOEuiopm           ;INVALID UNIT I/O PARAMETER
00000003        NOTLEGIT    EQU      IOEioreq           ;NOT LEGITIMATE CALL
00000032        INVTBLID    EQU      IOEtblid           ;INVALID TABLE ENTRY ID
00000033        TBLFULL     EQU      IOEtblfl           ;TIMER TABLE FULL
00000038        INVFNC      EQU      IOEfnccd           ;invalid function code
                ;
                ; MISCELLANEOUS EQUATES
                ;
00000006        UNMCMD      EQU     6                   ;UNMOUNT COMMAND CODE
00000004        ENABLEC     EQU     4                   ;ENABLE FUNCTION CODE
00000001        CARRYST     EQU     $01                 ;CARRY SET IN CCR
00000001        ON          EQU     1
00000000        OFF         EQU     0
```

```
; TIMER INTERRUPT SERVICE ROUTINE
; INTERNAL REGISTER USEAGE :
;               A0 - TEMP
;               A1 - TEMP
;               A2 - TIMER TABLE ADDRESS
;               A3 - ADDRESS OF CURRENT ENTRY'S FLAG'S LOW ORDER BYTE
;
;               D0 - TEMP
;               D1 - TEMP
;               D2 - INDEX TO CURRENT ENTRY IN TIMER TABLE
;
```

```
                     ;
                     ; TIMER INTERRUPT SERVICE ROUTINE
                     ; THIS ROUTINE IS INVOKED WHEN THE 50 MILLISECOND INTERVAL TIMER INTERRUPT
                     ; OCCURS.  IT CHECKS EACH ENTRY OF THE TIMER TABLE TO SEE IF IT'S USER SERVICE
                     ; ROUTINE SHOULD BE CALLED.
                     ;
0000                 TIMINT
0000 48E7 FFFE                  MOVEM.L   D0-D7/A0-A4,-(SP)         ;SAVE USER'S REGISTERS
0004 1039 0003 0F69             MOVE.B    T1CL.L,D0                 ;RESET VIA IFR T2 BIT
                     ;
                     ; for i := 1 to NUMENTS do
                     ;
000A 45FA 0524+                 LEA       TIMERTBL,A2               ;ADDRESS OF TIMER TABLE
000E 4282                       CLR.L     D2                        ;START WITH FIRST ENTRY
                     ;
                     ; IF HAVE VALID ENTRY THAT IS NOT DISABLE THEN SEE IF SHOULD CALL USER SERVICE ROUTINE
                     ;
0010 47F2 2001       TINEXT     LEA       TFLAGS+1(A2,D2.W),A3      ;ADDRESS OF CURRENT FLAGS + 1
0014 0813 0080                  BTST      #VALIDENT,(A3)            ;VALID ENTRY?
0018 6734                       BEQ.S     TICHKNXT                  ;NO, SEE IF ANOTHER ENTRY
001A 0813 0003                  BTST      #ENBLDSBL,(A3)            ;IS ENTRY ENABLED?
001E 662E                       BNE.S     TICHKNXT                  ;NO, SEE IF ANOTHER ENTRY
                     ;
                     ; GOT VALID ENTRY - TEST IF SHOULD CALL USER SERVICE ROUTINE
                     ;
0020 0472 0001 2008             SUBI.W    #1,TDWNCNT(A2,D2.W)       ;DOWN COUNT
0026 6626                       BNE.S     TICHKNXT                  ;NOT DONE, SEE IF ANOTHER ENTRY
                     ;
0028 48E7 2030                  MOVEM.L   D2/A2-A3,-(SP)            ;SAVE WORKING REGISTERS
002C 2872 200A                  MOVEA.L   REGA4(A2,D2.W),A4         ;SETUP USERS A4 AND A5
0030 2A72 200E                  MOVEA.L   REGA5(A2,D2.W),A5         ;REGISTERS
0034 2072 2002                  MOVEA.L   PTRUSRTN(A2,D2.W),A0      ;ADDRESS USER SERVICE ROUTINE
0038 4E90                       JSR       (A0)                      ;CALL USER SERVICE ROUTINE
003A 4CDF 0C04                  MOVEM.L   (SP)+,D2/A2-A3            ;RESTORE REGISTERS
                     ;
                     ; RESET DOWN COUNTER - ASSUME CONTINUOUS MODE
                     ;
003E 35B2 2006 2008             MOVE.W    TCOUNT(A2,D2.W),TDWNCNT(A2,D2.W)
                     ;
                     ; IF ENTRY IS IN 1 SHOT MODE THEN DELETE THE ENTRY.
                     ;
0044 0813 0001                  BTST      #CONT1SHT,(A3)            ;1 SHOT MODE?
0048 6704                       BEQ.S     TICHKNXT                  ;NO, SEE IF ANOTHER ENTRY
004A 0893 0000                  BCLR      #VALIDENT,(A3)            ;YES, DELETE ENTRY
                     ;
                     ; INCREMENT INDEX - IF NOT PAST END OF TABLE THEN DO NEXT ENTRY
                     ;
004E 0642 0012       TICHKNXT   ADDI.W    #TIMTLEN,D2               ;INDEX TO NEXT ENTRY
0052 0C42 00B4                  CMPI.W    #TABLELN,D2               ;PAST END OF TABLE?
0056 66B8                       BNE.S     TINEXT                    ;NO, DO NEXT ENTRY
                     ;
                     ; SEE IF SHOULD DO CONTEXT SWITCH
                     ;
                     ;          BSR.S     CHKCS                     ;RETURNS (A0) PTR TO SYSCOM
```

```
                           426* ;             BEQ.S      DOCS                ,DOES OWN EXIT(RTE)
                           427* ;
0058  4CDF  7FFF           428*               MOVEM.L    (SP)+,D0-D7/A0-A6    ;RESTORE USER REGISTERS
005C  4E73                 429* TUNRTE        RTE                             ;used by unitunmount
```

```
                  ;
                  ; DOCS - DO CONTEXT SWITCH
                  ;           ENTRY . MUST BE A GOTO CALL VIA A JUMP OR BRA NOT A SUBROUTINE
                  ;                   CALL. NO EXTRA STUFF ON STACK.
                  ,                   THE TOP OF STACK MUST BE THE USER'S REGISTERS
                  ,                   (A0) = POINTER TO SYSCOM
                  ;
005E 4228 005C    DOCS      CLR.B     SPNDFLG(A0)             ;CLEAR SUSPEND FLAG
                  ;
                  , SAVE EXISTING PROCESSES CONTEXT (PARTIAL, SCHEDULER DOES REST)
                  ;
0062 3028 0082              MOVE.W    CURPROC(A0),D0          ;GET CURRENT PROCESS #
0066 C0FC 00CC              MULU      #PTLEN,D0               ;CALC INDEX TO PROCESS TABLE ENTRY
006A 2268 00BB              MOVEA.L   PPTBL(A0),A1            ;ADDRESS OF PROCESS TABLE
006E 45F1 0000              LEA       0(A1,D0.W),A2           ;ADDRESS OF ENTRY
0072 720E                   MOVEQ     #NUMREGS-1,D1           ;COUNT OF POPS
                  ;
0074 24DF         DCSMOVR   MOVE.L    (SP)+,(A2)+             ;SAVE REGISTERS IN ENTRY
0076 51C9 FFFC              DBF       D1,DCSMOVR              ;IN ORDER D0-A6
                  ;
007A 355F 0040              MOVE.W    (SP)+,PTSR(A2)          ;SAVE SR AND PC OF CURRENT
007E 355F 003C              MOVE.W    (SP)+,PTPC(A2)          ;PROCESS
                  ;
                  ; CALL SCHEDULER VIA A FAKED RTE
                  ;
0082 2F28 00BB              MOVE.L    SCHEDPTR(A0),-(SP)      ;ENTRY POINT TO SCHEDULER
0086 3F3C 2700              MOVE.W    #SCHEDSR,-(SP)          ;SR FOR SCHEDULER
                  ;
008A 2868 00BB              MOVE.L    SCHDA4(A0),A4           ,SCHEDULER IS A PASCAL GLOBAL SUBROUTINE
008E 2A68 00BB              MOVE.L    SCHDA5(A0),A5           ;NEEDS ITS VALUES FOR A4 & A5
0092 4E73                   RTE
```

```
                        ;
                        ; CHKCS - SEE IF SHOULD DO A CONTEXT SWITCH
                        ;             EXIT : (NE) - DON'T DO CONTEXT SWITCH
                        ;                    (EQ) - DO SWITCH
                        ;                    (A0) = POINTER TO SYSCOM
                        ; IF (SUSPEND FLAG IS CLEAR) THEN DON'T DO SWITCH
                        ;
0094  2078  0180        CHKCS      MOVE.L     PSYSCOM.W,A0             ;ADDRESS OF SYSCOM
0098  4A28  005C                   TST.B      SPNDFLG(A0)              ;FLAG CLEAR?
009C  6706                         BEQ.S      CCSDONT                  ;YES
                        ;
                        ; IF (SUSPEND WAIT COUNTED SEMAPHORE = 0) THEN DO CONTEXT SWITCH
                        ;
009E  4A28  005A                   TST.B      SPWAITC(A0)
00A2  6002                         BRA.S      CCSEXIT
                        ;
00A4  7001              CCSDONT    MOVEQ      #1,D0                    ;FORCE DONT (NE)
00A6  4E75              CCSEXIT    RTS
```

```
;
; UNIT I/O PARAMETER PASSING DEFINITION
;
;   COMMAND        UNIT   ADDR   COUNT   BLOCK   MODE    IORESULT   BUSY
; 0 - INSTALL      D0.W                                  D7.W
; 1 - READ         D0.W   D1.L   D2.W            D5.W    D7.W
; 2 - WRITE        D0.W   D1.L   D2.W            D5.W    D7.W
; 3 - CLEAR        D0.W                                  D7.W
; 4 - BUSY         D0.W                                  D7.W       D6.W
; 5 - STATUS       D0.W   D1.L   D2.W                    D7.W
; 6 - UNMOUNT      D0.W                                  D7.W
;
; ALL REGISTER VALUES ON ENTRY ARE SAVED AND RESTORED EXCEPT D6 & D7.
; INTERNAL REGISTER USEAGE :
;             A0 - TEMP  (GLOBAL)
;             A1 - TEMP  (GLOBAL)
;             A2 - ADDRESS OF TIMER TABLE (GLOBAL)
;             A3 - ADDRESS OF USER'S BUFFER ADDRESS (GLOBAL)
;             A4 - ADDRESS OF INTERNAL FLAGS BYTE (BELL)
;             A5 - ADDRESS OF VIA SHIFT REGISTER (BELL)
;             A6 - ADDRESS OF VIA INTERRUPT ENABLE REGISTER (INSTALL)
;
;             D0 - TEMP
;             D1 - TEMP
;             D2 - COUNT OR CONTROL
;             D3 - TABLE ENTRY INDEX
;             D4 - PATTERN FOR BELL
;
```

```
                               ;
                                         GLOBAL      TIMERDRV
                               ;
                               ; TIMER DRIVER
                               ;
00A8                           TIMERDRV
00A8  6014                                 BRA.S       TIMR001                    ;*070782* JUMP AROUND HEADER
00AA  00                                   DATA.B      0                          ;DEVICE NOT BLOCKED
00AB  1F                                   DATA.B      31                         ;VALID CMDS - NOT UNITSTATUS
00AC  52 0A 16 00                          DATA.B      82,10,22,0                 ;DATE
00B0  0C                                   DATA.B      hmlen                      ;HEADER MSG LENGTH
00B1  54494D4552206472         xxx010      DATA.B      'TIMER driver'             ;HEADER MSG
00B9  69766572
      0000000C                 hmlen       EQU         %-xxx010
                               ;
00BE  0C44  0006               TIMR001     CMPI.W      #UNMCMD,D4                 ;VALID COMMAND
00C2  621C                                 BHI.S       TIMDERR                    ;NO
                               ;************ wait till change to D6 for busy return ************
                               ;****        MOVEM.L     D0-D5/A0-A6,-(SP)          ;SAVE REGISTERS
                               ;***********************************************************
00C4  48E7  7EFE                           movem.l     D1-D6/A0-A6,-(SP)          ;*** temp* for busy return in D0
00C8  4287                                 CLR.L       D7                         ;CLEAR IORESULT
00CA  2641                                 MOVEA.L     D1,A3                      ;ADDRESS OF USERS BUFFER
00CC  43FA  0018+                          LEA         TIMDTBL,A1                 ;TURN THE COMMAND INTO A
00D0  E34C                                 LSL.W       #1,D4                      ;INDEX TO THE FUNCTION
00D2  3831  4000                           MOVE.W      0(A1,D4.W),D4
00D6  4EB1  4000                           JSR         0(A1,D4.W)                 ;DO FUNCTION
                               ;************ wait till change to D6 for busy return ************
                               ;****        MOVEM.L     (SP)+,D0-D5/A0-A6          ;RESTORE SAVED REGISTER VALUES
                               ;***********************************************************
00DA  4CDF  7F7E                           movem.l     (SP)+,D1-D6/A0-A6          ;*** temp* for busy return in D0
00DE  4E75                                 RTS
                               ;
                               ; Invalid Command Error
                               ;
00E0  3E3C  0003               TIMDERR     MOVE.W      #NOTLEGIT,D7
00E4  4E75                                 RTS
                               ;
                               ; THE TIMER DRIVER JUMP TABLE
                               ;
00E6  027E                     TIMDTBL     DATA.W      TIMINST-TIMDTBL            ;UNITINISTALL
00E8  001A                                 DATA.W      TIMRD-TIMDTBL              ;UNITREAD
00EA  01D0                                 DATA.W      TIMWR-TIMDTBL              ;UNITWRITE
00EC  0014                                 DATA.W      TIMCLR-TIMDTBL             ;UNITCLEAR
00EE  000E                                 DATA.W      TIMBSY-TIMDTBL             ;UNITBUSY
00F0  02E6                                 DATA.W      TIMST-TIMDTBL              ;UNITSTATUS
00F2  02D4                                 DATA.W      TIMUNMT-TIMDTBL            ;UNITUNMOUNT
```

```
                          ;
                          ; TIMBSY - UNITRBUSY
                          ; BUSY FROM THE TIMER IS CURRENTLY UNDEFINED.
                          ;
00F4  3E3C  0003          TIMBSY     MOVE.W    #NOTLEGIT,D7
00F8  4E75                           RTS
                          ;
                          ; TIMCLR - UNITCLEAR
                          ; CLEAR THE TIMER IS CURRENTLY UNDEFINED.
                          ;
00FA  3E3C  0003          TIMCLR     MOVE.W    #NOTLEGIT,D7
00FE  4E75                           RTS
```

```
;
; TIMWR - UNITWRITE
; TIMRD - UNITREAD
;
; CODE FOR CALANDER CLOCK : IN INCLUDE FILE timer.clk.text
;
            INCLUDE    'TIMER.CLK.TEXT'
;
; file : timer.clk.text
; date : 22-OCTOBER-1982  kb
;
; FILE IS AN INCLUDE FILE FOR TIMER.DRV.TEXT , THE TIMER DRIVER.
; THIS IS THE UNITREAD AND UNITWRITE CODE FOR THE CALANDER CLOCK IN THE
; TIMER DRIVER.
;
; 04-23-82  kb  Changed IORESULT definitions to use the global file definitions
; 06-07-82  kb  Changed for new rev. 4 processor board changes, will find the
;                  correct address to use (either $30FE1 or $30F81)
; 07-07-82  kb  Changed error in RV3 and RV4 address equates, were reversed
; 09-20-82  lf  Changed write clock to stop/start clock in order to zero
;                  seconds and tenths fields
; 10-22-82  kb  Changed READCR so it reads 4 times with a MOVEP.L instruction.
;                  Checks to make sure the low order 3 bytes, all except the
,                  first read are the same.  If they are not it rereads the
;                  register.
;
```

```
                ;
                ; EQUATES FOR THE CLOCK ROUTINES
                ; PARAMTER BLOCK INDICES - RANGE FOR PARAMETER IN PARENTHESIS
                ;
00000000        DAYOFWK    EQU      0                  ;DAY OF THE WEEK (1-7)
00000002        MONTH      EQU      DAYOFWK+2          ;MONTH (1-12)
00000004        DAY        EQU      MONTH+2            ;DAY (1-31)
00000006        HOUR       EQU      DAY+2              ;HOURS (0-23)
00000008        MINS       EQU      HOUR+2             ;MINUTES (0-59)
0000000A        SECS       EQU      MINS+2             ;SECONDS (0-59)
0000000C        TENTHS     EQU      SECS+2             ;TENTHS OF SECONDS (0-9)
0000000E        LEAPYR     EQU      TENTHS+2           ;LEAP YEAR (0-3)
                ;
0000000E        LENPBR     EQU      TENTHS+2           ;LENGTH OF READ PARAMETER BLOCK
00000007        NUMRP      EQU      LENPBR/2           ;NUMBER OF READ PARAMETERS
                ;
00000010        LENPBW     EQU      LEAPYR+2           ;LENGTH OF WRITE PARAMETER BLOCK
00000008        NUMWP      EQU      LENPBW/2           ;NUMBER OF WRITE PARAMETERS
                ;
                ; CLOCK REGISTERS
                ;
00030F81        RV4ADDR    EQU      $30F81             ;SELECT/ADDRESS LATCH   **new board address
00030FE1        RV3ADDR    EQU      $30FE1             ;SELECT/ADDRESS LATCH  **old board address
00030D01        RWREG      EQU      $30D01             ;READ/WRITE CLOCK REGISTERS
0000000F        INTREG     EQU      15                 ;CLOCK INTERRUPT REG ADDR
0000000E        STRTSTOP   EQU      14                 ;START/STOP REGISTER
0000000D        LYREG      EQU      13                 ;LEAP YEAR REGISTER
00000001        TENTHSC    EQU       1                 ;TENTH OF SECONDS REGISTER (change 6/7)
                ;
0000000F        RDERR      EQU      $0F                ;REGISTER VALUE WHEN READ WHEN UPDATE
00000010        DSELCT     EQU      $10                ;DESELECT CHIP
                ;
                ; IORESULT CODES
                ;
00000037        PBLENER    EQU      IOEprmln           ;PARAMETER BLOCK WRONG LENGTH
00000039        CLOCKERR   EQU      IOEclkmf           ;CLOCK NOT WORKING
                ;
```

```
                     ;
                     ; TIMRD - UNITREAD OF CLOCK
                     ;
                     ; RETURN TO THE USER THE TIME IN THE REAL-TIME-CLOCK.
                     ; USER PASSES PARAMETER BLOCK POINTER IN D1 OF WHERE TO PUT THE TIME INFO.
                     ; TIME IS RETURNED IN BINARY, INTEGERS.   THE PARAMETER BLOCK HAS THE FORM .
                     ;
                     ; type ReadClockParameter = record
                     ;                           DayofWeek :integer;
                     ;                           Month : integer;
                     ;                           Day : integer;
                     ;                           Hour : integer,
                     ;                           Mins : integer,
                     ;                           Secs : integer,
                     ;                           Tenths : integer;
                     ;                         end;
                     ;
0100                 TIMRD
0100  2C41                     MOVE.L     D1,A6               ;A6 = PARAMETER BLOCK PTR
0102  2802                     MOVE.L     D2,D4               ;SAVE USER LENGTH
0104  6138                     BSR.S      RDCLOCK             ;READ THE CLOCK
0106  4A47                     TST.W      D7
0108  662C                     BNE.S      TRDEXIT             ,CAN'T READ CLOCK
                     ;
                     ; CONVERT AND PUT RESULT IN PARAMETER BLOCK
                     ;
010A  0C44  000E               CMPI.W     #LENPBR,D4          ;IS PARAMETER BLOCK LONG ENOUGH
010E  6528                     BCS.S      TRDERR              ;NO
0110  4BFA  03EC+              LEA        NUMBER,A5           ,# OF NIBBLES IN PARAMETER
0114  49FA  03DC+              LEA        DETAIL,A4           ;REGISTER ARRAY INDICES
0118  47FA  04DA+              LEA        REGARRAY,A3         ;REGISTER ARRAY
011C  4283                     CLR.L      D3                  ;INDEX INTO PARAM BLOCK
011E  4284                     CLR.L      D4                  ,INDEX INTO NUMBER ARRAY
0120  4286                     CLR.L      D6                  ,INDEX INTO DETAIL ARRAY
                     ;
0122  1035  4000     TRDGETP   MOVE.B     0(A5,D4.W),D0       ,# OF NIBBLES PARAMETER
0126  6158                     BSR.S      CVTOUT              ;CONVERT - RETURNS IN D1
0128  3D81  3000               MOVE.W     D1,0(A6,D3.W)       ,STORE PARAMETER
012C  5443                     ADDQ.W     #2,D3               .NEXT PARAMETER
012E  5204                     ADDQ.B     #1,D4               ;DO 7 PARAMETERS
0130  0C44  0007               CMPI.W     #NUMRP,D4
0134  66EC                     BNE.S      TRDGETP             ;DO AGAIN
0136  4E75           TRDEXIT   RTS
                     ;
                     ; ERROR - PARAMETER BLOCK THE WRONG LENGTH
                     ;
0138  3E3C  0037     TRDERR    MOVE.W     #PBLENER,D7
013C  4E75                     RTS
```

```
                            ;
                            ; RDCLOCK - READ ALL THE CLOCK REGISTERS INTO THE REGISTER ARRAY
                            ;
013E  6130                  RDCLOCK    BSR.S     LDADDR              ;GET CHIP ADDRESSES IN A0&A1
0140  7C07                             MOVEQ     #7,D6               ;COUNT OF FAILURES
                            ;
                            ; READ REGISTERS INTO ARRAY
                            ;
0142  5306                  RDCRST     SUBQ.B    #1,D6
0144  6B24                             BMI.S     RDCERR              ;RETRIED TO MANY TIMES
0146  7001                             MOVEQ     #1,D0               ;D0 = REGISTER #
0148  45FA  04AA+                      LEA       REGARRAY,A2
                            ;
014C  6166                  RDCREG     BSR.S     READCR              ;READ A SINGLE REGISTER REURNS IN D2
014E  14C2                             MOVE.B    D2,(A2)+            ;PUT IN ARRAY
0150  5200                             ADDQ.B    #1,D0
0152  0C00  000D                       CMPI.B    #RARDLEN+1,D0       ;IF DONE 12 TIMES STOP
0156  66F4                             BNE.S     RDCREG              ;DO AGAIN
                            ;
                            ; IF ANY REGISTER READ = $0F THEN READ REGISTERS WHEN TICKED AND MUST RESTART
                            ;
0158  700B                             MOVEQ     #RARDLEN-1,D0       ;DO 12 TIMES
015A  45FA  0498+                      LEA       REGARRAY,A2
                            ;
015E  0C1A  000F            RDDCHK     CMPI.B    #RDERR,(A2)+        ;BAD
0162  67DE                             BEQ.S     RDCRST              ;YES, REREAD CLOCK
0164  51C8  FFF8                       DBF       D0,RDDCHK
0168  4E75                             RTS
                            ;
                            ; ERROR - CHIP NOT WORKING
                            ;
016A  3E3C  0039            RDCERR     MOVE.W    #CLOCKERR,D7
016E  4E75                             RTS
                            ;
                            ; LDADDR - GET LATCH ADDRESS IN A0 AND R/W CLOCK ADDRESS IN A1
                            ;
0170  207A  047E+           LDADDR     MOVEA.L   ADDRREG,A0          ; GET saved address (change 6/7)
0174  2008                             MOVE.L    A0, D0              ; (change 6/7)
0176  67F2                             BEQ.S     RDCERR              ;NO CLOCK CHIP - ERROR EXIT (change 6/7)
0178  43F9  0003  0D01                 LEA       RWREG.L,A1
017E  4E75                             RTS
```

```
                 ;
                 ; CVOUT - CONVERT REGISTERS TO 1 PARAMETER
                 ;           ENTRY : (D0) = # OF REGISTERS TO USE (# OF DETAIL ELEMENTS)
                 ;                   DETAIL(D6) = REGISTER TO USE TO MAKE PARAMETER
                 ;                   (A4) = ADDRESS OF THE INDICES OF REGARRAY FOR EACH PARAMETER
                 ;                   (A3) = ADDRESS OF THE REGISTER ARRAY
                 ;           EXIT  : (D1) = PARAMETER CONVERTED
                 ;                   (D6) = UPDATED TO NEXT DETAIL ELEMENT FOR NEXT PARAMETER
                 ;
0180 4282          CVTOUT    CLR.L     D2
0182 4281                    CLR.L     D1
0184 1434 6000               MOVE.B    0(A4,D6.W),D2          ;GET REGARRAY INDEX
0188 5206                    ADDQ.B    #1,D6                  ;UPDATE INDEX
018A 1233 2000               MOVE.B    0(A3,D2.W),D1          ;FIRST NIBBLE
                 ;
018E 5300                    SUBQ.B    #1,D0                  ;IF NUMBER OF REGS=1  THEN
0190 670E                    BEQ.S     CVOEXIT                ;THEN DONE-IS VALID BINARY
                 ;
0192 E909                    LSL.B     #4,D1                  ;MOVE TO HI NIBBLE
0194 1434 6000               MOVE.B    0(A4,D6.W),D2          ;INDEX TO NEXT REGISTER
0198 5206                    ADDQ.B    #1,D6                  ;UPDATE INDEX
019A 8233 2000               OR.B      0(A3,D2.W),D1          ;PUT IN LOW NIBBLE
                 ;
019E 6102                    BSR.S     CBCDBIN                ;CONVERT BCD TO BINARY
01A0 4E75          CVOEXIT   RTS
```

```
                  ,
                  ; CBCDBIN - CONVERT 1 BYTE OF BCD (2 DIGITS) TO 1 BYTE OF BINARY
                  ;           ENTRY : (D1) = BCD BYTE OF DIGITS
                  ;           EXIT  . (D1) = BINARY BYTE
                  ;
01A2  4282          CBCDBIN   CLR.L     D2
01A4  1401                    MOVE.B    D1,D2          ,COPY OF BCD
01A6  0202  000F              ANDI.B    #$0F,D2        ;LOW ORDER DIGIT
01AA  E849                    LSR.W     #4,D1          ,MOVE OVER HI DIGIT
01AC  C2FC  000A              MULU      #10,D1         ,MAKE 10*DIGIT
01B0  D242                    ADD.W     D2,D1          ;MAKE FULL NUMBER
01B2  4E75                    RTS
```

```
                                  ;
                                  ; Changed routine on 10/22/82 kb
                                  ; READCR - READ CLOCK REGISTER
                                  ;       ENTRY : (A0) = LATCH ADDRESS
                                  ;               (A1) = R/W CLOCK ADDRESS
                                  ;               (D0) = REGISTER ADDRESS
                                  ,       EXIT  : (D2) = REGISTER VALUE READ
                                  ,
                                  ;  The movep.l instruction works because the I/O address space for the
                                  ;  clock is not fully decoded.
                                  ;
2*    01B4  48E7  0E00            READCR    MOVEM.L   D4-D6,-(SP)         ,save regs                      *kb 10/22/8

2*    01B8  6124                  RDCR10    BSR.S     SELREG              ;DESELECT THEN SELECT ADDRESS   *kb 10/22/8
2*    01BA  0549  0000                      MOVEP.L   0(A1),D2            ;read reg 4 times               *kb 10/22/8
      01BE  10BC  0010                      MOVE.B    #DSELCT,(A0)        ;DESELECT CHIP
2*    01C2  0282  0F0F  0F0F                ANDI.L    #$0F0F0F0F,D2       ;clear hi nibbles of all bytes  *kb 10/22/8
2*                                ;                                                                   *kb 10/22/8
2*                                ; make sure all bytes read are the same                             *kb 10/22/8
2*                                ;                                                                   *kb 10/22/8
2*    01C8  7801                            MOVEQ     #1, D4              ;ignore hi order byte, 1st read *kb 10/22/8
2*    01CA  2A02                            MOVE.L    D2,D5               ;save read value                *kb 10/22/8
2*    01CC  1C05                            MOVE.B    D5, D6              ;compare all to last read       *kb 10/22/8
2*    01CE                        RDCR20                                  ;                               *kb 10/22/8
2*    01CE  E08D                            LSR.L     #8, D5              ;chk next byte                  *kb 10/22/8
2*    01D0  BC05                            CMP.B     D5, D6              ;are they the same?             *kb 10/22/8
2*    01D2  66E4                            BNE.S     RDCR10              ,No, read reg again             *kb 10/22/8
2*    01D4  51CC  FFF8                      DBF       D4, RDCR20          ;Do until checked all 3 bytes   *kb 10/22/8
2*    01D8  4CDF  0070                      MOVEM.L   (SP)+,D4-D6         ;restore regs                   *kb 10/22/8
      01DC  4E75                            RTS
                                  ;
                                  , SELREG - DESELECT THEN SELECT CHIP REGISTER
                                  ;         ENTRY : (D0) = CLOCK REGISTER ADDRESS
                                  ,                 (A0) = LATCH ADDRESS
                                  ,
      01DE  7210                  SELREG    MOVEQ     #DSELCT,D1
      01E0  8200                            OR.B      D0,D1
      01E2  1081                            MOVE.B    D1,(A0)             ,DESELECT CHIP BY SETTING D4
      01E4  1080                            MOVE.B    D0,(A0)             ;SELECT ADDRESS
      01E6  4E75                            RTS
```

```
                        ;
                        ; INITCLK - PROCEDURE CALLED BY UNITINSTALL CODE (TIMINST) TO INITIALIZE
                        ;           THE CLOCK CHIP.
                        ;  ASSUMES THAT THE FOLLOWING CODE DOES NOT RESET THE CLOCK.???????
                        ;
01E8                    INITCLK
                        ;
                        ; START CLOCK - MUST DO IT FOR BOTH ADDRESSES
                        ;
01E8 43F9 0003 0D01             LEA     RWREG.L, A1             ;R/W ADDRESS           (change 6/7)
01EE 41F9 0003 0FE1             LEA     RV3ADDR.L, A0           ;DO OLD ADDRESS FIRST  (change 6/7)
01F4 612A                       BSR.S   STRTCLK                 ;                      (change 6/7)
01F6 41F9 0003 0F81             LEA     RV4ADDR.L, A0           ;DO NEW ADDRESS        (change 6/7)
01FC 6122                       BSR.S   STRTCLK                 ;                      (change 6/7)
                        ;                                                              (change 6/7)
                        ; FIND CORRECT ADDRESS OF THIS MACHINES PROCESSOR BOARD        (change 6/7)
                        ;                                                              (change 6/7)
01FE 6146                       BSR.S   FINDADDR                ;                      (change 6/7)
0200 6618                       BNE.S   INITEXIT                ;ERROR - NO CHIP       (change 6/7)
                        ;
                        ; INITIALIZE CHIP
                        ;
0202 6100 FF4C                  BSR     LDADDR                  ;GET CLOCK REGISTER ADDRESES
0206 4280                       CLR.L   D0                      ;REG ADDRESS
0208 4282                       CLR.L   D2                      ;DATA
020A 6100 0086                  BSR     WRITECR                 ;PUT IN NON-TEST MODE
                        ;
                        ; CLEAR INTERRUPTS
                        ;
020E 700F                       MOVEQ   #INTREG,D0              ;ADDRESS
0210 4282                       CLR.L   D2                      ;DATA
0212 617E                       BSR.S   WRITECR
0214 619E                       BSR.S   READCR                  ;READ 3 TIMES TO RESET
0216 619C                       BSR.S   READCR
0218 619A                       BSR.S   READCR
021A 4E75               INITEXIT RTS                            ;(change 6/7)
```

```
                          ;
                          ; STOPCLK - STOP CLOCK PROCEDURE                                          (09-20-82)
                          ;       ASSUMES A0 AND A1 ARE INITIALIZED.                                (09-20-82)
                          ;                                                                         (09-20-82)
021C  7400                STOPCLK     MOVEQ     #0,D2                      ;DATA                    (09-20-82)
021E  6002                            BRA.S     ST10                       ;                        (09-20-82)
                          ;                                                                         (09-20-82)
                          ; STRTCLK - START CLOCK PROCEDURE                                         (06-07-82)
                          ;       ASSUMES A0 AND A1 ARE INITIALIZED.                                (06-07-82)
                          ;                                                                         (06-07-82)
0220  7401                STRTCLK     MOVEQ     #1,D2                      ;DATA                    (06-07-82)
                                                                           ;                        (09-20-82)
0222  700E                ST10        MOVEQ     #STRTSTOP,D0               ;ADDRESS                 (09-20-82)
0224  616C                            BSR.S     WRITECR                    ;                        (06-07-82)
0226  4E75                            RTS                                  ;                        (06-07-82)
                          ;
                          ; RDTENTHS - read the tenths register of clock
                          ;       EXIT - (NC) = READ OK
                          ;              (C)  = ERROR - WRONG ADDRESS
                          ;              (D2) = REGISTER VALUE READ
                          ;
0228                      RDTENTHS
0228  7A03                            MOVEQ     #3,D5                      ;CHECK MAX. 4 TIMES FOR CLOCK TURNING

022A  7001                RDT10       MOVEQ     #TENTHSC,D0                ;read tenth of seconds register
022C  6186                            BSR.S     READCR
022E  0C02  000F                      CMPI.B    #RDERR, D2                 ;do until (no read error) or
0232  56CD  FFF6                      DBNE      D5, RDT10                  ; (tried 4 times)

0236  0C02  0009                      CMPI.B    #9, D2                     ;if not a BCD digit then wrong address
023A  6204                            BHI.S     RDTERR                     ;USE OTHER Address
023C  4285                            CLR.L     D5
023E  4E75                            RTS

0240  44FC  0001          RDTERR      MOVE.W    #CARRYST, CCR
0244  4E75                            RTS
```

```
                          ; FINDADDR - FIND ADDRESS OF CHIP'S ADDRESS LATCH. IT IS EITHER RV3ADDR OR RV4ADDR
                          ;            DEPENDING ON THE VERSION OF THIS PROCESSOR BOARD.
                          ; routine added with 6/7 change.
                          ;
0246                      FINDADDR
0246  47FA  03A8+                     LEA       ADDRREG, A3          ;WHERE TO SAVE CORRECT ADDRESS
024A  43F9  0003  0D01                LEA       RWREG.L,A1
0250  41F9  0003  0F81                LEA       RV4ADDR.L,A0         ;START WITH REV 4 ADDRESS
0256  7801                            MOVEQ     #1,D4                ;TRY ONLY TWO ADDRESSES

                          ; CHECK IF ADDRESS IN A0 IS CORRECT
                          ;
0258                      FA10
0258  61CE                            BSR.S     RDTENTHS             ;GET STARTING VALUE
025A  651E                            BCS.S     FANXT                ;WRONG CHIP TRY NEXT ADDRESS
025C  0C02  0009                      CMPI.B    #9, D2               ;WAIT UNTIL TENTH OF SECONDS
0260  6706                            BEQ.S     FAZERO               ;IS NEXT TENTH ==> D3.B
0262  1602                            MOVE.B    D2,D3
0264  5203                            ADDQ.B    #1,D3
0266  6002                            BRA.S     FA20
0268  4283                FAZERO      CLR.L     D3
026A  3C3C  7530          FA20        MOVE.W    #30000,D6            ;MUST READ AT LEAST TENTH SEC.

                          ; READ TENTHS UNTIL IT CHANGES OR UNTIL IT TRIED TO LONG
                          ;
026E                      FA30
026E  61B8                            BSR.S     RDTENTHS             ;GET NEXT VALUE
0270  6508                            BCS.S     FANXT                ;WRONG CHIP TRY NEXT ADDRESS
0272  B602                            CMP.B     D2,D3                ;HAS TIME TICKED
0274  6716                            BEQ.S     FAFNDIT              ;YES, FOUND CORRECT ADDRESS
0276  51CE  FFF6                      DBF       D6, FA30             ;READ AGAIN
                          ;
                          ; NOT THIS ADDRESS TRY OTHER ADDRESS
                          ;
027A  41F9  0003  0FE1    FANXT       LEA       RV3ADDR.L, A0
0280  51CC  FFD6                      DBF       D4, FA10

                          ; ERROR - NEITHER ADDRESS WORKED
                          ;
0284  4293                            CLR.L     (A3)                 ;SHOW NO CHIP ADDRESS
0286  3E3C  0039                      MOVE.W    #CLOCKERR, D7
028A  4E75                            RTS

                          ; FOUND CORRECT ADDRESS
                          ;
028C  2688                FAFNDIT     MOVE.L    A0, (A3)
028E  4287                            CLR.L     D7
0290  4E75                            RTS
```

```
                      ;
                      ;  WRITECR - WRITE A CLOCK REGISTER
                      ;           ENTRY : (D0) = REGISTER ADDRESS
                      ;                   (D2) = DATA
                      ;
0292  6100  FF4A      WRITECR    BSR       SELREG              ,DESELECT THEN SELECT REG.
0296  1282                       MOVE.B    D2,(A1)             ,WRITE DATA
0298  10BC  0010                 MOVE.B    #DSELCT,(A0)        ;DESELECT CHIP
029C  4E75                       RTS
                      ;
                      ; WRITEREGS - WRITE THE CLOCK REGISTERS FROM THE REGISTER ARRAY
                      ;
029E  6100  FED0      WRITEREGS  BSR       LDADDR              ;GET CHIP ADDRESSES
                      ;
                      ; WRITE REGISTERS
                      ;
02A2  7001                       MOVEQ     #1,D0               ;REGISTER ADDRESS
02A4  45FA  034E+                LEA       REGARRAY,A2
                      ,
02A8  141A            WRONER     MOVE.B    (A2)+,D2            ;REGISTER DATA
02AA  61E6                       BSR.S     WRITECR             ,WRITE DATA
02AC  5200                       ADDQ.B    #1,D0               ;NEXT REGISTER ADDRESS
02AE  0C00  000E                 CMPI.B    #STRTSTOP,D0        ,STOP AT START/STOP REG.
02B2  66F4                       BNE.S     WRONER
02B4  4E75                       RTS
```

```
                  ;
                  ; TIMWR - SET CLOCK FROM PARAMETER BLOCK
                  ;             PARAMETER BLOCK FOR UNITWRITE :
                  ;
                  ; type WriteClockParameter = record                      range
                  ;                              DayofWeek : integer;       1-7
                  ;                              Month : integer;           1-12
                  ;                              Day : integer;             1-31
                  ;                              Hour : integer;            0-23
                  ;                              Mins : integer;            0-59
                  ;                              Secs : integer;            0-59
                  ;                              Tenths : integer;          0-9
                  ;                              LeapYear : integer;        0-3
                  ;                             end;
                  ;
02B6  2641        TIMWR      MOVE.L    D1,A3              ;ADDRESS OF PARAMETER BLOCK
                  ;
                  ; PROCESS BINARY PARAMETERS
                  ;
02B8  611A                   BSR.S     VALBIN             ;VALIDATE PARAMS
02BA  6512                   BCS.S     TWRERR             ;NO GOOD
02BC  613C                   BSR.S     CVTBINR            ;CONVERT BINARY TO BCD OF REGISTERS
                  ;
                  ; ZERO SECONDS AND TENTHS OF SECONDS                        (09-20-82)
                  ;                                                           (09-20-82)
02BE  6100 FE80              BSR       LDADDR             ; GET CLOCK ADDRESSES (09-20-82)
02C2  6100 FF58              BSR       STOPCLK            ;                     (09-20-82)
02C6  6100 FF58              BSR       STRTCLK            ;                     (09-20-82)
                  ;
                  ; WRITE OUT REGISTER ARRAY
                  ;
02CA  61D2                   BSR.S     WRITEREGS
02CC  4E75                   RTS
                  ;
                  ; ERROR - INVALID CLOCK PARAMETER
                  ;
02CE  3E3C 0036   TWRERR     MOVE.W    #INVPRM,D7
02D2  4E75                   RTS
```

```
                      ;
                      ; VALBIN - VALIDATE BINARY PARAMETER BLOCK
                      ;           ENTRY : (A3) = ADDRESS OF PARAMETER BLOCK
                      ;           EXIT  : (NC) = GOOD PARAMETERS
                      ;                   (C)  = ERROR, OUT OF RANGE
                      ;
02D4 284B             VALBIN   MOVEA.L  A3,A4            ;SAVE PB ADDRESS
02D6 7807                      MOVEQ    #NUMWP-1,D4      ;DO ALL 8 PARAMETERS
02D8 4BFA 0246+                LEA      RANGES,A5        ;LIST OF PARAMETER RANGES(BYTES)
                      ;
                      ; COMPARE EACH PARAMETER TO IT'S LOW AND HI RANGE VALUE
                      ;
02DC 4280             VBCHK    CLR.L    D0
02DE 101D                      MOVE.B   (A5)+,D0         ;GET LOW BOUND RANGE
02E0 B054                      CMP.W    (A4),D0          ;PARAM>=LOW BOUND THEN OK
02E2 6210                      BHI.S    VBERR            ;ERROR, TO LOW
02E4 101D                      MOVE.B   (A5)+,D0         ;GET HI BOUND RANGE VALUE
02E6 B054                      CMP.W    (A4),D0          ;PARAM<=HI BOUND THEN OK
02E8 650A                      BCS.S    VBERR            ;ERROR, TO HIGH
02EA 548C                      ADDQ.L   #2,A4            ;NEXT PARAMETER LOW BYTE
02EC 51CC FFEE                 DBF      D4,VBCHK
02F0 4280                      CLR.L    D0               ;SHOW NO ERROR
02F2 4E75                      RTS
                      ;
                      ; ERROR EXIT - OUT OF RANGE
                      ;
02F4 44FC 0001        VBERR    MOVE.W   #CARRYST,CCR     ;SHOW ERROR
02F8 4E75                      RTS
```

```
                    ;
                    ; CVTBINR - CONVERT VALID PARAMETER BLOCK FROM BINARY INTO REGARRAY BCD NIBLES
                    ;            ENTRY : (A3) = ADDRESS OF PARAMETER BLOCK
                    ;
02FA 49FA 0306+     CVTBINR   LEA      HI,A4                  ;HI NIBBLE HOLD ARRAY
02FE 4BFA 030A+               LEA      LOW,A5                 ;LOW NIBBLE HOLD ARRAY
0302 7807                     MOVEQ    #NUMWP-1,D4            ;FOR i := 8 to 1 do
0304 760F                     MOVEQ    #LEAPYR+1,D3           ;INDEX TO PARAMETER
                    ;
0306 4280           CBRNIBS   CLR.L    D0
0308 1033 3000                MOVE.B   0(A3,D3.W),D0          ;GET PARAMETER[i]
030C 2200                     MOVE.L   D0,D1
030E 82FC 000A                DIVU     #10,D1                 ;HI NIBBLE := PARAM DIV 10
0312 2001                     MOVE.L   D1,D0                  ;LOW IS REMAINDER FROM DIV
0314 4840                     SWAP     D0                     ;LOW := PARAMETER-(HI*10)
0316 1981 4000                MOVE.B   D1,0(A4,D4.W)          ;SAVE HI
031A 1B80 4000                MOVE.B   D0,0(A5,D4.W)          ;SAVE LOW
031E 5543                     SUBQ.W   #2,D3                  ;NEXT PARAMETER INDEX
0320 51CC FFE4                DBF      D4,CBRNIBS             ;DOWNTO 0
                    ;
                    ; SETUP REGISTER ARRAY
                    ;
0324 4283                     CLR.L    D3                     ;REMOVE GARBAGE
0326 47FA 02CC+               LEA      REGARRAY,A3
032A 780C                     MOVEQ    #RAWRLEN-1,D4          ;MOVE TO 13 REGISTERS
032C 45FA 01E5+               LEA      NIBBLE,A2              ;WHICH NIBBLE FOR THIS REG.
0330 43FA 01D4+               LEA      INREGB,A1              ;WHICH PARAMETER IS REG FROM
                    ;
0334 1631 4000      CBRREGS   MOVE.B   0(A1,D4.W),D3          ;INDEX TO HI & LOW FOR THIS REG.
0338 1035 3000                MOVE.B   0(A5,D3.W),D0          ;ASSUME LOW NIBBLE
033C 4A32 4000                TST.B    0(A2,D4.W)             ;IF 0 THEN USE LOW NIBBLE
0340 6704                     BEQ.S    CBRULOW                ;IS LOW
0342 1034 3000                MOVE.B   0(A4,D3.W),D0          ;ELSE GET HI NIBBLE
0346 1780 4000      CBRULOW   MOVE.B   D0,0(A3,D4.W)          ;PUT NIBBLE IN REGISTER HOLD
034A 51CC FFE8                DBF      D4,CBRREGS
                    ;
034E 6102                     BSR.S    CVTLPYR                ;CONVERT LEAP YEAR REG
0350 4E75                     RTS
```

```
                    ;
                    ; CVTLPYR - CONVERT LEAP YEAR PARAMETER TO THE REGISTER VALUE FOR THE
                    ;           CLOCK CHIP.  8,4,2,1 WHERE 8 IS FOR LEAP YEAR AND THE OTHER NUMBERS
                    ;           ARE FOR THE YEARS AFTER THE LEAP YEAR.  THEREFORE, 4 IS LEAP YEAR+1,
                    ;           2 IS LEAP YEAR+2, 1 IS LEAP YEAR+3.
                    ;
0352 102B 000C      CVTLPYR     MOVE.B    LYREG-1(A3),D0          ;LEAP YEAR REG. = 13(INDEX=12)
0356 7203                       MOVEQ     #3,D1                   ;D0 IS PARAMETER (RANGE 0-3)
0358 9200                       SUB.B     D0,D1                   ;CALCULATE WHICH BIT TO SET
035A 4280                       CLR.L     D0                      ;BIT# := 3-PARAMETER
035C 03C0                       BSET      D1,D0                   ;D0 IS LEAP YEAR VALUE
035E 1740 000C                  MOVE.B    D0,LYREG-1(A3)          ,PUT IN REGISTER ARRAY
0362 4E75                       RTS
```

```
                          ;
                          ; TIMINST - UNITINSTALL
                          ; INSTALL THE TIMER INTERRUPT ROUTINE AND SET UP THE VIA
                          ;
0364                      TIMINST
                          ;
0364 4DF9 0003 0F7D                   LEA       IER.L,A6
036A 1CBC 007F                        MOVE.B    #DISABL,(A6)            ;TURN OFF ALL INTERUPTS ON VIA
                          ;
                          ; INITIALIZE TIMER TABLE
                          ;
036E 41FA 01C0+                       LEA       TIMERTBL,A0             ;ADDRESS OF TIMER TABLE
0372 43FA 0270+                       LEA       TIMERTBL+(TIMTLEN*NUMENTS),A1 ;1ST BYTE AFTER TABLE
                          ;
0376 30BC 0000            TINST10     MOVE.W    #0,(A0)                 ;CLEAR FLAGS OF EACH ENTRY
037A D0FC 0012                        ADDA.W    #TIMTLEN,A0             ;POINT AT NEXT ENTRY
037E B3C8                             CMPA.L    A0,A1                   ;AT END OF TABLE
0380 66F4                             BNE.S     TINST10                 ;NO
                          ;
                          ; PUT ADDRESS OF INTERRUPT ROUTINE IN VECTOR
                          ;
0382 41FA FC7C+                       LEA       TIMINT,A0
0386 21C8 0074                        MOVE.L    A0,VECTOR.W
                          ;
                          ; SETUP VIA
                          ;
038A 13FC 0040 0003                   MOVE.B    #ACRBYTE,ACR.L          ;FREE RUN MODE PB7 OUTPUT DISABLED
0390 0F77
0392 13FC 0050 0003                   MOVE.B    #TIMEL,T1LL.L           ;TIMER #1 LATCH LOW
0398 0F6D
039A 13FC 00C3 0003                   MOVE.B    #TIMEH,T1LH.L           ;TIMER #1 LATCH HIGH
03A0 0F6F
03A2 13FC 00C3 0003                   MOVE.B    #TIMEH,T1CH.L           ;TIMER #1 COUNTER HIGH - FORCE LOAD
03A8 0F6B
03AA 13FC 00FF 0003                   MOVE.B    #CLEAR,IFR.L            ;CLEAR IFR
03B0 0F7B
                          ;
                          ; ENABLE TIMER #2
                          ;
03B2 1CBC 00C0                        MOVE.B    #ENBLT1,(A6)            ;TURN INTERUPTS ON FOR T1
                          ;
                          ; INITIALIZE CLOCK - SOURCE IN TIMER.CLK.TEXT INCLUDE FILE
                          ;
03B6 6000 FE30                        BRA       INITCLK                 ;DOES RETURN WHEN INITCLK DOES
```

```
                         ;
                         ; TIMUNMT - UNITUNMOUNT
                         ; TURN OFF THE VIA INTERRUPTS AND POINT THE TIMER INTERRUPT VECTOR AT A RTE.
                         ;
03BA                     TIMUNMT
03BA  13FC  007F  0003                MOVE.B    #DISABL,IER.L          ,TURN OFF ALL INTERUPTS ON VIA
03C0  0F7D
03C2  41FA  FC98+                     LEA       TUNRTE,A0              ,WITH TIMER INTERRUPT CODE
03C6  21C8  0074                      MOVE.L    A0,VECTOR.W            ,POINT VECTOR AT RTE
03CA  4E75                            RTS
```

```
                         ;
                         ; TIMST - UNITSTATUS
                         ; THIS PPROCEDURE CONTAINS THE BELL ROUTINE AND THE 4 TIMER TABLE MANIPULATION
                         ; PROCEDURES, CREATE, DELETE, DISABLE, AND ENABLE.
                         ;
                         ;           ENTRY    D2 - CONTROL CODE USED TO SELECT FUNCTIONS
                         ;                    A3 - BUFFER ADDRESS = PTR TO PARAMETER BLOCK
                         ;
03CC  0C42  0004         TIMST       CMPI.W   #ENABLEC,D2            ;VALID FUNCTION CODE
03D0  6212                           BHI.S    TSTERR                 ;NO
                         ;
03D2  45FA  015C+                    LEA      TIMERTBL,A2            ;ADDRESS OF TIMER TABLE
03D6  43FA  0012+                    LEA      TSTTBL,A1              ;TURN THE CONTROL CODE INTO A
03DA  E34A                           LSL.W    #1,D2                  ;INDEX TO THE FUNCTION
03DC  3431  2000                     MOVE.W   0(A1,D2.W),D2
03E0  4EF1  2000                     JMP      0(A1,D2.W)             ;DO FUNCTION
                         ;
                         ; Invalid Function Code Error
                         ;
03E4  3E3C  0038         TSTERR      MOVE.W   #INVFNC,D7
03E8  4E75                           RTS
                         ;
                         ; THE TIMER DRIVER JUMP TABLE
                         ;
03EA  00A0               TSTTBL      DATA.W   TSTBELL-TSTTBL         ;BELL
03EC  000A                           DATA.W   TSTCRE8-TSTTBL         ;CREATE TABLE ENTRY
03EE  004E                           DATA.W   TSTDELT-TSTTBL         ;DELETE TABLE ENTRY
03F0  0060                           DATA.W   TSTDSBL-TSTTBL         ;DISABLE TABLE ENTRY
03F2  0072                           DATA.W   TSTENBL-TSTTBL         ;ENABLE TABLE ENTRY
```

```
                        ;
                        ; TSTCRE8 - CREATE TABLE ENTRY
                        ;           ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                        ;                   A2 = ADDRESS OF TIMER TABLE
                        ;                   A4 = VALUE WHEN TIMER DRIVER CALLED
                        ;                   A5 = VALUE WHEN TIMER DRIVER CALLED
                        ;           PARAMETER BLOCK .
                        ;              1) ADDRESS OF USER SERVICE ROUTINE TO INSTALL IN ENTRY (LONGWORD)
                        ;              2) COUNT OF 50 MILLISECOND PERIODS TO WAIT (WORD)
                        ;              3) FLAGS (WORD) -
                        ;                        bit D1 = CONTINUOUS/1SHOT MODE FLAG
                        ;                        bit D2 = SKIP FIRST CALL FLAG
                        ;              4) RETURN SPACE FOR TABLE ENTRY ID, THE ENTRY NUMBER (WORD)
                        ;
03F4 4280               TSTCRE8     CLR.L      D0                         ;ENTRY #
03F6 4283                           CLR.L      D3                         ;ENTRY INDEX
                        ;
                        ; FIND AN UNUSED ENTRY IF ONE AVAILABLE
                        ;
03F8 0832 0000 3001     TCRCKNXT    BTST       #VALIDENT,TFLAGS+1(A2,D3.W)
03FE 6712                           BEQ.S      TCRFOUND                   ;FOUND ONE
0400 0643 0012                      ADDI.W     #TIMTLEN,D3                ;ELSE SEE IF AT END OF TABLE
0404 5240                           ADDQ.W     #1,D0                      ;NEXT ENTRY NUMBER
0406 0C40 000A                      CMPI.W     #NUMENTS,D0                ;IN TABLE?
040A 66EC                           BNE.S      TCRCKNXT                   ;YES
                        ;
                        ; ERROR TABLE FULL
                        ;
040C 3E3C 0033                      MOVE.W     #TBLFULL,D7
0410 6024                           BRA.S      TCREXIT
                        ;
                        ; FOUND UNUSED ENTRY - SET IT UP
                        ;
0412 259B 3002          TCRFOUND    MOVE.L     (A3)+,PTRUSRTN(A2,D3.W)    ;PUT IN USER SERVICE RTN ADDRES
0416 3593 3006                      MOVE.W     (A3),TCOUNT(A2,D3.W)       ;COUNT OF 50 MS. TICKS
041A 359B 3008                      MOVE.W     (A3)+,TDWNCNT(A2,D3.W)     ;SET DOWN COUNTER
041E 321B                           MOVE.W     (A3)+,D1                   ;GET FLAGS
0420 08C1 0000                      BSET       #VALIDENT,D1               ;SHOW ENTRY IN USE
0424 0881 0003                      BCLR       #ENBLDSBL,D1               ;SHOW ENABLED
0428 3581 3000                      MOVE.W     D1,TFLAGS(A2,D3.W)         ;PUT IN ENTRY
042C 258C 300A                      MOVE.L     A4,REGA4(A2,D3.W)          ;SAVE USERS A4 AND A5 REGISTERS
0430 258D 300E                      MOVE.L     A5,REGA5(A2,D3.W)
                        ;
                        ; RETURN TO USER TABLE ENTRY ID (THE ENTRY NUMBER)
                        ;
0434 3680                           MOVE.W     D0,(A3)
0436 4E75               TCREXIT     RTS
```

```
                         ;
                         ; TSTDELT - DELETE TABLE ENTRY
                         ;          ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                         ;                  A2 = ADDRESS OF TIMER TABLE
                         ;          PARAMETER BLOCK :
                         ;             1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                         ;
0438                     TSTDELT                                        ;GET INDEX TO ENTRY
0438  613A                          BSR.S     VALIDID                   ;IS ID VALID?
043A  6508                          BCS.S     TDELERR                   ;INVALID-ERROR EXIT
                         ;
                         ; VALID ENTRY INDEX - DELETE ENTRY
                         ;
043C  08B2  0060  3001              BCLR      #VALIDENT,TFLAGS+1(A2,D3.W)
0442  6004                          BRA.S     TDELEXIT
                         ;
                         ; INVALID TABLE ID ERROR
                         ;
0444  3E3C  0032         TDELERR    MOVE.W    #INVTBLID,D7
0448  4E75               TDELEXIT   RTS
```

```
                               ;
                               ; TSTDSBL - DISABLE TABLE ENTRY
                               ;          ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                               ;                  A2 = ADDRESS OF TIMER TABLE
                               ;          PARAMETER BLOCK :
                               ;             1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                               ;
044A                           TSTDSBL                                          ;GET INDEX TO ENTRY
044A  6128                                BSR.S      VALIDID                    ;IS ID VALID?
044C  6508                                BCS.S      TDSBERR                    ;INVALID-ERROR EXIT
                               ;
                               ; VALID ENTRY INDEX - DISABLE ENTRY
                               ;
044E  08F2  0003  3001                    BSET       #ENBLDSBL,TFLAGS+1(A2,D3.W)
0454  6004                                BRA.S      TDSBEXIT
                               ;
                               ; INVALID TABLE ID ERROR
                               ;
0456  3E3C  0032               TDSBERR    MOVE.W     #INVTBLID,D7
045A  4E75                     TDSBEXIT   RTS
```

```
                  ;
                  ; TSTENBL - ENABLE TABLE ENTRY
                  ;           ENTRY . A3 = ADDRESS OF PARAMETER BLOCK
                  ;                   A2 = ADDRESS OF TIMER TABLE
                  ;           PARAMETER BLOCK :
                  ;              1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                  ;
045C              TSTENBL                                          ;GET INDEX TO ENTRY
045C  6116                   BSR.S     VALIDID                     ;IS ID VALID?
045E  650E                   BCS.S     TENBERR                     ;INVALID-ERROR EXIT
                  ;
                  ; VALID ENTRY INDEX - ENABLE ENTRY AND RESTART DOWN COUNTER
                  ;
0460  35B2  3006  3008       MOVE.W    TCOUNT(A2,D3.W),TDWNCNT(A2,D3.W)
0464  08B2  0003  3001       BCLR      #ENBLDSBL,TFLAGS+1(A2,D3.W)
046C  6004                   BRA.S     TENBEXIT
                  ;
                  ; INVALID TABLE ID ERROR
                  ;
046E  3E3C  0032  TENBERR    MOVE.W    #INVTBLID,D7
0472  4E75        TENBEXIT   RTS
                  ;
                  ; VALIDID - VALIDATE TABLE ENTRY ID IN PARAMETER
                  ;           ENTRY . A3 = ADDRESS OF PARAMETER BLOCX
                  ;           EXIT  . D3 = TABLE INDEX
                  ;                   (C) = INVALID TABLE ENTRY ID
                  ;                   (NC)= VALID TABLE ENTRY ID
                  ;
0474  3613        VALIDID    MOVE.W    (A3),D3                     ;GET TABLE ENTRY ID
                  ;
                  ; TABLE ENTRY ID IS THE ENTRY NUMBER - MAKE SURE LESS THAN NUMBER OF ENTRIES IN TABLE
                  ;
0476  0C43  0009             CMPI.W    #NUMENTS-1,D3               ;IS INDEX LESS THAN TABLELN?
047A  6306                   BLS.S     VALCALC                     ;YES, CALCULATE INDEX
                  ;
047C  44FC  0001  VALERR     MOVE.W    #CARRYST,CCR                ;SHOW ERROR
0480  6006                   BRA.S     VALEXIT
                  ;
                  ; HAVE VALID TABLE ENTRY
                  ;
0482  C6FC  0012  VALCALC    MULU      #TIMTLEN,D3                 ;CALCULATE INDEX
0486  4280                   CLR.L     D0                          ;CLEAR CARRY
0488  4E75        VALEXIT    RTS
```

```
                        , TSTBELL - BELL ROUTINE
                        ,           ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                        ;           PARAMETER BLOCK :
                        ;             1) FREQUENCY (WORD)
                        ;             2) SPEAKER ON/OFF PATTERN (BYTE)
                        ;             3) FILLER (BYTE)
                        ;             4) DURATION IN 50 MILLISECOND PERIODS (WORD)
                        ; INIT VIA FOR FREQUENCY W/O DISTURBING TIMER #1
                        ;
048A 4BF9 0003 0F75     TSTBELL     LEA      SHIFTREG.L,A5
0490 1ABC 0000                      MOVE.B   #0,(A5)             ;TURNOFF BELL FOR SURE
0494 0039 0010 0003                 ORI.B    #RUNT2,ACR.L        ;SET TIMER #2 AS COUNT DOWWN
049A 0F77
049C 613A                           BSR.S    SETT2               ;PUT FREQUENCY IN TIMER
049E 49FA 0144+                     LEA      IFLAGS,A4
04A2 0894 0000                      BCLR     #SHUTOFF,(A4)       ;PUT TIMER ON
                        ;
                        , CALL CREATE TO SETUP ONE SHOT INTERVAL TIMER CALL
                        ;
04A6 43FA 013E+                     LEA      IPRMBLK,A1          ,ADDR OF INTERNAL PARM BLOCK
04AA 2F0B                           MOVE.L   A3,-(SP)            ;SAVE PARAMETER BLOCK ADDRESS
04AC 302B 0004                      MOVE.W   DURATN(A3),D0       ;SAVE THE COUNT
04B0 2649                           MOVEA.L  A1,A3               ;CREATE EXPECTS PRM BLK ADR IN A3
04B2 41FA 0034+                     LEA      BELSRVR,A0          ;BELL SERVICE ROUTINE ADDRESS
04B6 22C8                           MOVE.L   A0,(A1)+            ,PUT IN PARAMETER BLOCK
04B8 3280                           MOVE.W   D0,(A1)             ;PUT IN COUNT
04BA 6100 FF38                      BSR      TSTCRE8             ;CALL CREATE
04BE 265F                           MOVE.L   (SP)+,A3            ,BELL PARAMETER BLOCK ADDRESS
                        ;
04C0 1AAB 0002                      MOVE.B   PATTERN(A3),(A5)    ;TURN ON BELL
                        ;
                        ; WAIT FOR SHUT OFF
                        ;
04C4 0814 0000          TBELWAIT    BTST     #SHUTOFF,(A4)       ;DONE?
04C8 67FA                           BEQ.S    TBELWAIT            ;NO
                        ;
                        ; DONE SHUT OFF TIMER #2 AND BELL
                        ;
04CA 1ABC 0000          TBELDONE    MOVE.B   #0,(A5)             ,CLEAR SHIFT REG TO SHUT OFF BELL
04CE 0239 00EF 0003                 ANDI.B   #STOPT2,ACR.L
04D4 0F77
04D6 4E75               TBELEXIT    RTS
```

```
                              ;
                              ; SETT2 - SET TIMER #2 TO FREQUENCY IN PARAMETER BLOCK
                              ;
04D8  13EB  0001  0003        SETT2      MOVE.B     1(A3),T2LL.L          ;SET LATCH
04DE  0F71
04E0  13D3  0003  0F73                   MOVE.B     (A3),T2CH.L           ;SET COUNTER AND CLEAR IFR T2 FLAG
04E4  4E75                               RTS
                              ;
                              ; BELL TIMER SERVICE ROUTINE
                              ;
04E8  41FA  00FA+             BELSRVR    LEA        IFLAGS,A0             ;TELL BELL ROUTINE DONE
04EC  08D0  0000                         BSET       #SHUTOFF,(A0)         ;& TO SHUT OFF SPEAKER AND
04F0  4E75                               RTS                              ;TIMER #2
```

```
                        1345* ;
                        1346* ,
                        1347* , DATA AREA
                        1348* ; CONSTANTS FOR CALANDER CLOCK
                        1349* ; CONVERSION ARRAYS BCD TO BINARY/REGISTER TO PARAMETER BLOCK CONVERSION
                        1350* ;
04F2  09 0B 0A 08 07 06 1351* DETAIL      DATA.B     9,11,10,8,7,6,5,4,3,2,1,0 ,REGISTERS WHICH MAKE THE PARAMETERS
04F8  05 04 03 02 01 00
04FE  01 02 02 02 02 02 1352* NUMBER      DATA.B     1,2,2,2,2,2,1             ,# OF REGISTERS FOR PARAMEETER
0504  01
0505  00                1353*             DATA.B     0                         ;**** FILL *****
                        1354* ;
                        1355* ; CONVERSION ARRAYS FOR PARAMETER BLOCK TO REGISTER ARRAY CONVERSION
                        1356* ;
0506  06 05 05 04 04 03 1357* INREGB      DATA.B     6,5,5,4,4,3,3,2,2,0,1,1,7 ;WHICH PARAM IN REG[i] (BINARY)
050C  03 02 02 00 01 01
0512  07
0513  00 00 01 00 01 00 1358* NIBBLE      DATA.B     0,0,1,0,1,0,1,0,1,0,0,1,0 ,WHICH NIBBLE- 1=HI
0519  01 00 01 00 00 01
051F  00
                        1359* ;
                        1360* ; RANGE VALUES FOR CLOCK PARAMTER BLOCK FIELDS, 1 BYTE LOW, 1 BYTE HI FOR EACH
                        1361* ;  OF 8 PARAMTER BLOCK FIELDS
                        1362* ;
0520  01 07 01 0C 01 1F 1363* RANGES      DATA.B     1,7,1,12,1,31,0,23,0,59,0,59,0,9,0,3
0526  00 17 00 3B 00 3B
052C  00 09 00 03
                        1364* ;
                        1365* ; VARIABLE DATA AREA
                        1366* ;
                        1367* ; THE TIMER TABLE - 10 ENTRIES
                        1368* ;
0530  0000  0000  0000  1369* TIMERTBL    DATA.W     0,0,0,0,0,0,0,0,0         ; ENTRY # 0
0536  0000  0000  0000
053C  0000  0000  0000
      00000012          1370* TIMTLEN     EQU        %-TIMERTBL                ,length of entry
0542  0000  0000  0000  1371*             DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 1
0548  0000  0000  0000
054E  0000  0000  0000
0554  0000  0000  0000  1372*             DATA.W     0,0,0,0,0,0,0,0,0         ; ENTRY # 2
055A  0000  0000  0000
0560  0000  0000  0000
0566  0000  0000  0000  1373*             DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 3
056C  0000  0000  0000
0572  0000  0000  0000
0578  0000  0000  0000  1374*             DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 4
057E  0000  0000  0000
0584  0000  0000  0000
058A  0000  0000  0000  1375*             DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 5
0590  0000  0000  0000
0596  0000  0000  0000
059C  0000  0000  0000  1376*             DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 6
05A2  0000  0000  0000
05A8  0000  0000  0000
```

```
05AE  0000  0000  0000  1377*            DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 7
05B4  0000  0000  0000
05BA  0000  0000  0000
05C0  0000  0000  0000  1378*            DATA.W     0,0,0,0,0,0,0,0,0         ; ENTRY # 8
05C6  0000  0000  0000
05CC  0000  0000  0000
05D2  0000  0000  0000  1379*            DATA.W     0,0,0,0,0,0,0,0,0         , ENTRY # 9
05D8  0000  0000  0000
05DE  0000  0000  0000
      00000084          1380* TABLELN    EQU        %-TIMERTBL                ,length of table in bytes
      0000000A          1381* NUMENTS    EQU        TABLELN/TIMTLEN           ;# of entries in table
                        1382* ,
                        1383* , INTERNAL FLAGS AND PARAMETER BLOCK
                        1384* ,
05E4  0000              1385* IFLAGS     DATA.W     0                         ,USE ONLY 1ST BYTE
05E6  0000  0000  0000  1386* IPRMBLK    DATA.W     0,0,0,2,0                 ,BELL USES FOR CREATE CALL
05EC  0002  0000
                        1387* , THE ADDRESS AND COUNT ARE SET IN THE BELL ROUTINE - FLAGS ARE ALWAYS
                        1388* , ONE-SHOT MODE ONLY
                        1389* ,
                        1390* , CLOCK DATA AREA
                        1391* , CLOCK ADDRESS AND SELECT LATCH ADDRESS SAVE AREA
                        1392* ,
05F0  00000000          1393* ADDRREG    DATA.L      0                       ,(CHANGE 6/7)
                        1394*
                        1395* , REGISTER ARRAY HOLD
                        1396* ,
05F4  00 00 00 00 00 00 1397* REGARRAY   DATA.B     0,0,0,0,0,0,0,0,0,0,0,0
05FA  00 00 00 00 00 00
      0000000C          1398* RARDLEN    EQU        %-REGARRAY                ,NUMBER OF REGISTERS READ
0600  00                1399*            DATA.B     0
      0000000D          1400* RAWRLEN    EQU        %-REGARRAY                ;NUMBER OF REGISTERS WRITTEN
0601  00                1401*            DATA.B     0                         ;FILL
                        1402* ,
                        1403* , NIBBLE HOLD FOR PARAMETER TO REGISTER CONVERSION
                        1404* ,
0602  00 00 00 00 00 00 1405* HI         DATA.B     0,0,0,0,0,0,0,0
0608  00 00
060A  00 00 00 00 00 00 1406* LOW        DATA.B     0,0,0,0,0,0,0,0
0610  00 00
                        1407* ,
      000000A8+         1408*            END        TIMERDRV

 ACR       00030F77  BITD6     00000006  CLOCKERR 00000039  CURPROC   00000002  DOCS        00005E+
 ACRBYTE   00000040  BITD7     00000007  CLRSC    00000006  CURSON    00000002  DSELCT    00000010
 ACTIVE    00000006  CARRYST   00000001  CONT1SHT 00000001  CVOEXIT     0001A0+ DURATN    00000004
 ADDRREG     0005F0+ CBCDBIN     0001A2+ CSATTR1  00000010  CVTBINR     0002FA+ ENABLEC   00000004
 BELSRVR     0004E8+ CBRNIBS     000306+ CSATTR2  00000011  CVTLPYR     000352+ ENBLDSBL  00000003
 BITD0     00000000  CBRREGS     000334+ CSBPCH   00000006  CVTOUT      000180+ ENBLT1    000000C0
 BITD1     00000001  CBRULOW     000346+ CSFRSTCH 00000008  DAY       00000004  FA10        000258+
 BITD2     00000002  CCSDONT     0000A4+ CSLASTCH 0000000A  DAYOFWK   00000008  FA20        00026A+
 BITD3     00000003  CCSEXIT     0000A6+ CSLPCH   00000004  DCSMOVR     000074+ FA30        00026E+
 BITD4     00000004  CHKCS       000094+ CSMASK   0000000C  DETAIL      0004F2+ FAFNDIT     00028C+
 BITD5     00000005  CLEAR     000000FF  CSTBLLOC 00000000  DISABL    0000007F  FANXT       00027A+
FAZERO     000268+  MMBTBLK   0000001A  SCCODEJT 00000022  SVBLKIO  0000002C  TENBEXIT   000472+
FINDADDR   000246+  MMBTDEV   00000012  SCCURRK  00000048  SVCLI    0000007C  TENTHS   0000000C
FREQ     00000000   MMBTDRV   00000018  SCCURRW  00000044  SVCLOSE  00000020  TENTHSC  00000001
GRAPHIC  00000001   MMBTSLT   00000014  SCDEVTAB 00000014  SVCRKPTH 00000060  TFLAGS   00000000
HI         000602+  MMBTSRV   00000016  SCDIRNAM 00000018  SVDELENT 00000090  TICHKNXT   00004E+
HMLEN    0000000C   MMBTSW    00000010  SCFREEHP 00000004  SVDSPOSE 00000038  TIMBSY     0000F4+
HOUR     00000006   MMHICOD   0000000C  SCHDA4   000000BB  SVFLPDIR 00000088  TIMCLR     0000FA+
IER      00030F7D   MMHIDTA   00000004  SCHDA5   000000BB  SVGET    00000014  TIMDERR    0000E0+
IFLAGS     0005E4+  MMLOCOD   00000008  SCHEDPTR 000000BB  SVGETDIR 00000048  TIMDTBL    0000E6+
IFR      00030F7B   MMLODTA   00000000  SCHEDSR  00002700  SVGETVNM 00000080  TIME     0000C350
INITCLK    0001E8+  MONTH     00000002  SCIORSLT 00000000  SVINIT   00000018  TIMEH    000000C3
INITEXIT   00021A+  NIBBLE      000513+ SCJTABLE 00000008  SVMARK   0000003C  TIMEL    00000050
INREGB     000506+  NOAUTOLF  00000004  SCMEMMAP 00000032  SVMAVAIL 00000044  *TIMERDRV   0000A8+
INSMOD   00000002   NOSCROLL  00000005  SCNUMPRO 00000028  SVNEW    00000034  TIMERTBL   000530+
INTREG   0000000F   NOTLEGIT  00000003  SCNXTPRO 00000026  SVOPEN   0000001C  TIMINST    000364+
INVCURS  00000003   NUMBER      0004FE+ SCPROCNO 00000002  SVPUT    00000010  TIMINT     000000+
INVFNC   00000038   NUMENTS   0000000A  SCPROTBL 0000002A  SVPUTDIR 00000094  TIMR001    0000BE+
INVPRM   00000036   NUMREGS   0000000F  SCROOTW  00000040  SVRDCHAR 00000028  TIMRD      000100+
INVRSE   00000000   NUMRP     00000007  SCSLTTBL 0000003C  SVRLEASE 00000040  TIMST      0003CC+
INVTBLID 00000032   NUMWP     00000008  SCSUSINH 0000005A  SVSCHDIR 0000008C  TIMTLEN  00000012
IOECLKMF 00000039   OFF       00000000  SCSUSREQ 0000005C  SVSEEK   00000030  TIMUNMT    0003BA+
IOEFNCCD 00000038   ON        00000001  SCSYSIN  00000010  SVUBUSY  0000000C  TIMWR      0002B6+
IOEIOREQ 00000003   PATTERN   00000002  SCSYSOUT 0000000C  SVUCLEAR 00000008  TINEXT     000010+
IOEKYBTE 00000035   PBLENER   00000037  SCTODAY  00000020  SVUINSTL 00000098  TINST10    000376+
IOENFDRV 0000002D   PPTBL     000000BB  SCUSERID 0000004C  SVUREAD  00000004  TRDERR     000138+
IOENOBUF 00000017   PSYSCOM   00000180  SCUTABLE 0000001C  SVUSTAT  00000064  TRDEXIT    000136+
IOENODSP 00000028   PTLEN     000000CC  SCVRSDAT 00000052  SVUWRITE 00000000  TRDGETP    000122+
IOENOKYB 00000029   PTPC      0000003C  SCVRSNBR 0000004E  SVVALDIR 00000084  TSTBELL    00048A+
IOENOOMN 0000002B   PTRUSRTN  00000002  SCWNDTBL 00000056  SVWRCHAR 00000024  TSTCRE8    0003F4+
IOENOPRT 0000002C   PTSR      00000040  SECS     0000000A  SYSBYTES 00000186  TSTDELT    000438+
IOENOTIM 0000002A   RANGES      000520+ SELREG     0001DE+ SYSKYBDF 00000184  TSTDSBL    00044A+
IOENOTRN 00000015   RARDLEN   0000000C  SETT2      0004D8+ SYSWIN   00000005  TSTENBL    00045C+
IOEPRMLN 00000037   RAWRLEN   0000000D  SHIFTREG 00030F75  T1CH     00030F6B  TSTERR     0003E4+
IOETBLFL 00000033   RDCERR      00016A+ SHUTOFF  00000000  T1CL     00030F69  TSTTBL     0003EA+
IOETBLID 00000032   RDCLOCK     00013E+ SKIP1ST  00000002  T1LH     00030F6F  TUNRTE     00005C+
IOETBLIU 00000034   RDCR10      0001B8+ SPNDFLG  0000005C  T1LL     00030F6D  TWRERR     0002CE+
IOETIMOT 00000016   RDCR20      0001CE+ SPWAITC  0000005A  T2CH     00030F73  UNDSCR   00000001
IOEUIOPM 00000036   RDCREG      00014C+ ST10       000222+ T2INT    00000020  UNMCMD   00000006
IOEWNDBE 00000021   RDCRST      000142+ STACSLT  00000004  T2LL     00030F71  UTBLF    00000006
IOEWNDCS 00000022   RDDCHK      00015E+ STACSRV  00000006  TABLELN  000000B4  UTBLK    0000001C
IOEWNDDC 00000023   RDERR     0000000F  STALSLT  00000008  TBELDONE   0004CA+ UTDID    00000008
IOEWNDDS 00000024   RDT10       00022A+ STALSRV  0000000A  TBELEXIT   0004D6+ UTDRV    00000016
IOEWNDFN 00000020   RDTENTHS    000228+ STBTSLT  00000000  TBELWAIT   0004C4+ UTIODRV  00000002
IOEWNDIW 00000025   RDTERR      000240+ STBTSRV  00000002  TBLFULL  00000033  UTLEN    00000020
IOEWNDWN 00000027   READCR      0001B4+ STINFO   0000000C  TCOUNT   00000006  UTMTD    00000007
IOEWNDWR 00000026   REGA4     0000000A  STINFOL  00000004  TCRCKNXT   0003F8+ UTRO     0000001A
IPRMBLK    0005E6+  REGA5     0000000E  STNDRV   00000002  TCREXIT    000436+ UTSIZ    00000010
LDADDR     000170+  REGARRAY    0005F4+ STNMBR   00000008  TCRFOUND   000412+ UTSLT    00000014
LEAPYR   0000000E   RUNT2     00000010  STOPCLK    00021C+ TDELERR    000444+ UTSPT    00000018
LENPBR   0000000E   RV3ADDR   00030FE1  STOPT2   000000EF  TDELEXIT   000448+ UTSRV    00000015
LENPBW   00000010   RV4ADDR   00030F81  STRTCLK    000220+ TDSBERR    000456+ UTTPS    00000019
LOW        00060A+  RWREG     00030D01  STRTSTOP 0000000E  TDSBEXIT   00045A+ UTTYP    00000017
LYREG    0000000D   SCBOOTDV  00000036  STTYPE   00000001  TDWNCNT  00000006  VALBIN     0002D4+
MINS     00000008   SCBOOTNM  0000002E  SUSPEND  00000007  TENBERR    00046E+ VALCALC    000482+
VALERR     00047C+  VERT      00000000  WRBASEY  00000010  WRGRORGY 0000001E  WRLNGTHY 00000014
VALEXIT    000488+  VIDDEFLT 00000003  WRBITOFS 0000001A  WRHOMEOF 0000000C  WRONER     0002A8+
VALIDENT 00000000   VIDSET    00000007  WRCHARPT 00000000  WRHOMEPT 00000004  WRRCDLEN 00000023
VALIDID    000474+  WRAPON    00000004  WRCURADR 00000008  WRITECR    000292+  WRSTATE  00000022
VBCHK      0002DC+  WRATTR1   00000020  WRCURSX  00000016  WRITEREG   00029E+  XXX010     0000B1+
VBERR      0002F4+  WRATTR2   00000021  WRCURSY  00000018  WRLENGTH 00000024
VECTOR   00000074   WRBASEX   0000000E  WRGRORGX 0000001C  WRLNGTHX 00000012
```

0 errors   1409 lines.

```
ASM68K Cross Reference  0.01d     07 Sep 1982          Date. Nov 1, 1982
Cross-Reference Listing
File ID: TIMER.DRV.TEXT

ACR         303* 1103  1302  1329
ACRBYTE     316* 1103
ACTIVE      232*
ADDRREG     721   886  1393*
BELSRVR    1313  1341*
BITD0       262*  286   299
BITD1       263*  287
BITD2       264*  288
BITD3       265*  289
BITD4       266*
BITD5       267*
BITD6       268*
BITD7       269*
CARRYST     356*  878  1021  1282
CBCDBIN     750   758*
CBRNIBS    1033* 1042
CBRREGS    1052* 1058
CBRULOW    1055  1057*
CCR         878  1021  1282
CCSDONT     472   479*
CCSEXIT     477   480*
CHKCS       470*
CLEAR       321* 1107
CLOCKERR    633*  716   921
CLRSC       243*
CONT1SHT    287*  413
CSATTR1     204*
CSATTR2     206*
CSBPCH      200*
CSFRSTCH    201*
CSLASTCH    202*
CSLPCH      199*
CSMASK      203*
CSTBLLOC    198*
CURPROC     331*  442
CURSON      239*
CVOEXIT     743   751*
CVTBINR     977  1028*
CVTLPYR    1060  1069*
CVTOUT      672   736*
DAY         604*  605
DAYOFWK     602*  603
DCSMOVR     448*  449
DETAIL      665  1351*
DISABL      319* 1084  1122
DOCS        438*
DSELCT      628*  781   801   937
DURATN      295* 1311
ENABLEC     355* 1135
ENBLDSBL    289*  392  1195  1239  1261
ENBLT1      320* 1111
FA10        893*  916
FA20        900   902*
```

```
FA30       906*  911
FAFNDIT    910   926*
FANXT      895   908   915*
FAZERO     897   901*
FINDADDR   824   885*
FREQ       293*
GRAPHIC    238*
HI        1028  1405*
HMLEN      521   523*
HOUR       605*  606
IER        304* 1083  1122
IFLAGS    1304  1341  1385*
IFR        305* 1107
INITCLK    812* 1115
INITEXIT   825   842*
INREGB    1050  1357*
INSMOD     228*
INTREG     622*  836
INVCURS    240*
INVFNC     350* 1146
INVPRM     346*  992
INVRSE     226*
INVTBLID   348* 1223  1244  1266
IOECLKMF    66*  633
IOEFNCCD    65*  350
IOEIOREQ    37*  347
IOEKYBTE    62*
IOENFDRV    57*
IOENOBUF    41*
IOENODSP    52*
IOENOKYB    53*
IOENOOMN    55*
IOENOPRT    56*
IOENOTIM    54*
IOENOTRN    39*
IOEPRMLN    64*  632
IOETBLFL    60*  349
IOETBLID    59*  348
IOETBLIU    61*
IOETIMOT    40*
IOEUIOPM    63*  346
IOEWNDBE    44*
IOEWNDCS    45*
IOEWNDDC    46*
IOEWNDDS    47*
IOEWNDFN    43*
IOEWNDIW    48*
IOEWNDWN    50*
IOEWNDWR    49*
IPRMBLK   1309  1386*
LDADDR     688   721*  829   942   981
LEAPYR     609*  614  1031
LENPBR     611*  612   642
LENPBW     614*  615
LOW       1029  1406*
LYREG      624* 1069  1074
MINS       606*  607
MMBTBLK    156*
```

```
MMBTDEV    152*
MMBTDRV    155*
MMBTSLT    153*
MMBTSRV    154*
MMBTSW     151*
MMHICOD    150*
MMHIDTA    148*
MMLOCOD    149*
MMLODTA    147*
MONTH      603*  604
NIBBLE    1049  1358*
NOAUTOLF   230*
NOSCROLL   242*
NOTLEGIT   347*  545   562   568
NUMBER     664  1352*
NUMENTS   1089  1180  1279  1381*
NUMREGS    338*  339   446
NUMRP      612*  676
NUMWP      615* 1002  1030
OFF         23   358*
ON         253   357*
PATTERN    294* 1319
PBLENER    632*  682
PPTBL      332*  444
PSYSCOM     71*  470
PTLEN      337*  443
PTPC       339*  340   452
PTRUSRTN   278*  403  1190
PTSR       340*  451
RANGES    1083  1363*
RARDLEN    701   706  1398*
RAWRLEN   1048  1400*
RDCERR     694   716*  723
RDCLOCK    656   688*
RDCR10     779*  792
RDCR20     789*  793
RDCREG     698*  702
RDCRST     693*  710
RDDCHK     709*  711
RDERR      627*  709   870
RDT10      868*  871
RDTENTHS   865*  894   907
RDTERR     874   878*
READCR     698   777*  839   840   841   869
REGA4      281*  282   401  1197
REGA5      282*  402  1198
REGARRAY   666   696   707   947  1047  1397* 1398  1400
RUNT2      317* 1302
RV3ADDR    620*  817   915
RV4ADDR    619*  819   888
RWREG      621*  724   816   887
SCBOOTDV    94*
SCBOOTNM    92*
SCCODEJT    88*
SCCURRK    102*
SCCURRW    101*
SCDEVTAB    84*
SCDIRNAM    85*
```

```
SCFREEHP    80*
SCHDA4     334*  459
SCHDA5     335*  460
SCHEDPTR   333*  456
SCHEDSR    342*  457
SCIORSLT    78*
SCJTABLE    81*
SCMEMMAP    93*
SCNUMPRO    90*
SCNXTPRO    89*
SCPROCNO    79*  331
SCPROTBL    91*
SCROOTW    100*
SCSLTTBL    99*
SCSUSINH   107*  330
SCSUSREQ   108*  329
SCSYSIN     83*
SCSYSOUT    82*
SCTODAY     87*
SCUSERID   103*
SCUTABLE    86*
SCVRSDAT   105*
SCVRSNBR   104*
SCWNDTBL   106*
SECS       607*  608
SELREC     779   801*  935
SETT2     1303  1335*
SHIFTREG   312* 1300
SHUTOFF    299* 1305  1323  1342
SKIP1ST    288*
SPNDFLG    329*  438   471
SPWAITC    330*  476
ST10       849   856*
STACSLT    183*
STACSRV    184*
STALSLT    185*
STALSRV    186*
STBTSLT    181*
STBTSRV    182*
STINFO     187*
STINFOL    192*
STNDRV     191*
STNMBR     189*
STOPCLK    848*  982
STOPT2     318* 1329
STRTCLK    818   820   854*  983
STRTSTOP   623*  856   952
STTYPE     190*
SUSPEND    233*
SVBLKIO    125*
SVCLI      135*
SVCLOSE    122*
SVCRKPTH   133*
SVDELENT   140*
SVDSPOSE   128*
SVFLPDIR   138*
SVGET      119*
SVGETDIR   132*
```

```
SVGETVNM  136*
SVINIT    120*
SVMARK    129*
SVMAVAIL  131*
SVNEW     127*
SVOPEN    121*
SVPUT     118*
SVPUTDIR  141*
SVRDCHAR  124*
SVRLEASE  130*
SVSCHDIR  139*
SVSEEK    126*
SVUBUSY   117*
SVUCLEAR  116*
SVUINSTL  142*
SVUREAD   115*
SVUSTAT   134*
SVUWRITE  114*
SVVALDIR  137*
SVWRCHAR  123*
SYSBYTES   73*
SYSKYBDF   72*
SYSWIN    231*
T1CH      309* 1106
T1CL      308*  380
T1LH      307* 1105
T1LL      306* 1104
T2CH      311* 1336
T2INT     322*
T2LL      310* 1335
TABLELN   420  1380* 1381
TBELDONE 1328*
TBELEXIT 1330*
TBELWAIT 1323* 1324
TBLFULL   349* 1185
TCOUNT    279*  409  1191  1260
TCRCKNXT 1176* 1181
TCREXIT  1186  1203*
TCRFOUND 1177  1190*
TDELERR  1214  1223*
TDELEXIT 1219  1224*
TDSBERR  1235  1244*
TDSBEXIT 1240  1245*
TOWNCNT   280*  281   397   409  1192  1260
TENBERR  1256  1266*
TENBEXIT 1262  1267*
TENTHS    608*  609   611
TENTHSC   615*  868
TFLAGS    277*  389  1176  1196  1218  1239  1261
TICHKNXT  391   393   398   414   419*
TIMBSY    554   562*
TIMCLR    553   568*
TIMDERR   526   545*
TIMDTBL   533   550*  550   551   552   553   554   555   556
TIME      323*  324   325
TIMEH     324*  325  1105  1106
TIMEL     325* 1104
TIMERDRV  512   516* 1408
```

```
TIMERTBL   384  1088  1089  1138  1369* 1370  1380
TIMINST    550  1081*
TIMINT     378* 1098
TIMRCG1    517   525*
TIMRD      551   653*
TIMST      555  1135*
TIMTLEN    419  1089  1092  1178  1287  1370* 1381
TIMUNMT    556  1121*
TIMWR      552   971*
TINEXT     389*  421
TINST10   1091* 1094
TRDERR     643   682*
TRDEXIT    658   678*
TRDGETP    671*  677
TSTBELL   1151  1300*
TSTCREB   1152  1171* 1316
TSTDELT   1153  1212*
TSTDSBL   1154  1233*
TSTENBL   1155  1254*
TSTERR    1136  1146*
TSTTBL    1139  1151* 1151  1152  1153  1154  1155
TUNRTE     429* 1123
TWRERR     976   992*
UNDSCR     227*
UNMCMD     354*  525
UTBLF      163*
UTBLK      175*
UTDID      165*
UTDRV      169*
UTIODRV    162*
UTLEN      176*
UTMTD      164*
UTRO       173*
UTSIZ      166*
UTSLT      167*
UTSPT      171*
UTSRV      168*
UTTPS      172*
UTTYP      170*
VALBIN     975  1001*
VALCALC   1280  1287*
VALZRR    1282*
VALEXIT   1283  1289*
VALIDENT   286*  390   415  1176  1194  1218
VALIDID   1213  1234  1255  1275*
VBCHK     1307* 1015
VBERR     1010  1013  1021*
VECTOR     273* 1099  1124
VERT       237*
VIDDEFLT   229*
VIDSET     244*
WRAPON     241*
WRATTR1    224*
WRATTR2    235*
WRBASEX    215*
WRBASEY    216*
WRBITOFS   221*
WRCHARPT   211*
```

```
WRCURADR   213*
WRCURSX    219*
WRCURSY    220*
WRGRORGX   222*
WRGRORGY   223*
WRHOMEOF   214*
WRHOMEPT   212*
WRITECR    832   838   857   935*  950
WRITEREG   942*  987
WRLENGTH   249*
WRLNGTHX   217*
WRLNGTHY   218*
WRONER     949*  953
WRRCDLEN   247*
WRSTATE    246*
XXX010     522*  523
```

```
; file : drv.dtacom.text
; date : 09-Feb-1983
;
; This is the datacom driver source
;
;     date          by         rev level            comments
;______________________________________________________________
;
;   10/12/82       rpk          1           initial version    no protocols-nothin
;   10/21/82       rpk          2           added auto line feed flag on writes
;   10/27/82       rpk          3           MADE BUSY ONLY TELL ABOUT READ BUFFER
;   11/29/82       kb           4a          Started additions for protocols and
;                                             compatibility with old printer driver
;                                             unitstatus interface.
;   01/05/83       kb           4e          changed auto line feed flag useage
;   01/04/83       kb           4f          added switch of water marks when switch back
;                                             to default read buffer.
;   01/12/83       kb                       fixed bug in FINDLIN routine, using
;                                             wrong register for bit test. changed
;                                             bit number register from D0 to D5.
;   02/09/83       kb                       added setup of UART base reg in DCTLINT
;                                              in ChkLines
;
;***************************************************************************
;
; INCLUDE FILES USED :
;       /ccos/os.gbl.asm.text                    ;OS GLOBAL EQUATES
;       dcom.equ.text                            ;definitions for driver
;
; INCLUDE OS GLOBALS HERE
            LIST       1
            INCLUDE    'DCOM.EQU.TEXT'
```

```
                    , Equates for Data com driver
                    ; file . DCOM.EQU.TEXT
                    , date : 24 - January - 1983
                    ,
                    ;added definition of LFsprsflg as byte value of UCSD and Apple p-systems
                    ,       mode flags (D5) : 1-24-83  kb
                    ,
                    ,
                    ; EQUATES FOR ALL DATACOM DRIVER SOFTWARE
                    , BIT NUMBER DEFINITIONS
                    ;
00000000            BITD0       EQU         0                       ,BIT 0
00000001            BITD1       EQU         1                       ;BIT 1
00000002            BITD2       EQU         2                       ,BIT 2
00000003            BITD3       EQU         3                       ,BIT 3
00000004            BITD4       EQU         4                       ,BIT 4
00000005            BITD5       EQU         5                       ,BIT 5
00000006            BITD6       EQU         6                       ,BIT 6
00000007            BITD7       EQU         7                       ;BIT 7
                    ,
                    ; Flags for port common Flag word
                    ,
00000000            PORTFLG     EQU         BITD0                   ,IF 0 THEN INIT PORT 0 ELSE INIT PORT 1
                    ,
                    ; Buffer control table INTERNAL Flag  bit definitions **LO BYTE***  BF_INTL
                    , Low order byte
                    ,
00000004            ENQFLG      EQU         BITD4                   ,SENT ENQ WAITING FOR ACK
                    ,
                    , Buffer Control Table PROTOCOL flag bit definitions  ****lo byte**** BF_PROF
                    ,
00000000            LINE        EQU         BITD0                   ,LINE TYPE HANDSHAKE
00000001            XONXOFF     EQU         BITD1                   ;XON/XOFF HANDSHAKE
00000002            ENQACK      EQU         BITD2                   ,ENQ/ACK HANDSHAKE
00000003            CTSLIN      EQU         BITD3                   ;LINE IS CTS
00000004            DSRLIN      EQU         BITD4                   ,LINE IS DSR
00000005            DCDLIN      EQU         BITD5                   ,LINE IS DCD
00000006            INVBUSY     EQU         BITD6                   .1=LINE IS INVERTED(0) WHEN BUSY
00000007            ETXACK      EQU         BITD7                   ,ETX/ACK HANDSHAKE
                    ,
                    ; BUFFER CONTROL TABLE PROTOCOL FLAG BIT DEFINITIONS ***HI BYTE***BF_PROF
                    ,
00000000            PROT_P2     EQU         BITD0                   ,IF SET THEN SOME TYPE OF PROTOCOL EXISTS
                                                                    ,ELSE NO PROTOCOLS --BUFFERS OVERFLOW ETC
00000001            MODM_P2     EQU         BITD1                   ,IF SET THEN A MODEM PROTOCOL EXISTS
00000002            NMOD_P2     EQU         BITD2                   ;IF SET THEN NULL MODEM PROTOCOL(PROBABLY OF
LITTLE USE)
00000003            FULL_P2     EQU         BITD3                   ,IF SET THEN FULL DUPLEX (DEFAULT)
                    ,                                               ,OTHERWISE HALF DUPLEX
                    ;
                    ;
                    ;       WRITE BUFFER flag word bit definitions    FLAG 1 ->>lo byte
                    ;
00000000            BUSY_W1     EQU         BITD0                   ,WRITE BUSY FLAG
00000001            ERR_W1      EQU         BITD1                   ,BUFFER SIZE ERROR FOUND IN INIT INT RTN
```

```
00000002        ALTBF_W1    EQU         BITD2                   ;IF SET(1) HAVE ALTERNATE BUFFER TO USE
00000004        OUTE_W1     EQU         BITD4                   ;IF 0 DATA FROM BUFFER TO PORT ENABLED
00000005        INPE_W1     EQU         BITD5                   ;IF 0 DATA FROM USER TO BUFFER ENABLED
00000006        OUTC_W1     EQU         BITD6                   ;IF SET(1) THEN USER IS CONTROLLING OUTE
                                                                ;OTHERWISE CONTROLLED INTERNALLY
00000007        INPC_W1     EQU         BITD7                   ;IF SET(1) THEN USER IS CONTROLLING INPE
                ;
                ;      WRITE BUFFER flag word bit definitions   FLAG 2 ->>lo byte
                ;
00000000        FULL_W2     EQU         BITD0                   ;IF SET (1) THEN BUFFER IS FULL
00000001        EMPT_W2     EQU         BITD1                   ;IF SET (1) THEN BUFFER IS EMPTY
00000002        LOST_W2     EQU         BITD2                   ;DATA LOST ON INPUT (USER OVERRUNS BUFFER)
00000003        SNDLF_W2    EQU         BITD3                   ;IF SET THEN SEND AN LF
00000004        AULF_W2     EQU         BITD4                   ;IF SET THEN always send a LF after a CR
                ;
                ;       READ BUFFER flag word bit definitions   FLAG 1 ->>LO BYTE
                ;
00000000        BUSY_R1     EQU         BITD0                   ;READ BUSY FLAG
00000001        ERR_R1      EQU         BITD1                   ;UART ERROR FLAG
00000002        ALTBF_R1    EQU         BITD2                   ;IF SET(1) HAVE ALTERNATE BUFFER TO USE
00000004        OUTE_R1     EQU         BITD4                   ;IF 0 DATA FROM BUFFER TO USER ENABLED
00000005        INPE_R1     EQU         BITD5                   ;IF 0 DATA FROM PORT TO BUFFER ENABLED
00000006        OUTC_R1     EQU         BITD6                   ;IF SET(1) THEN USER IS CONTROLLING OUTE
                                                                ;OTHERWISE CONTROLLED INTERNALLY
00000007        INPC_R1     EQU         BITD7                   ;IF SET(1) THEN USER IS CONTROLLING INPE
                ;
                ;       READ BUFFER flag word bit definitions   FLAG 2 LO BYTE
                ;
00000000        FULL_R2     EQU         BITD0                   ;IF SET (1) THEN BUFFER IS FULL
00000001        EMPT_R2     EQU         BITD1                   ;IF SET (1) THEN BUFFER IS EMPTY
00000002        LOST_R2     EQU         BITD2                   ;DATA LOST ON INPUT (PORT OVERRUNS BUFFER)
                ;
                ;       CONTROL CHARACTER BUFFER flag word bit definitions  LO BYTE
                ;
00000000        FULL_CB     EQU         BITD0                   ;IF SET (1) THEN BUFFER IS FULL
00000001        EMPT_CB     EQU         BITD1                   ;IF SET (1) THEN BUFFER IS EMPTY
                ;
                ; 68000 Interrupt Auto Vector Addresses
                ;
00000064        VEC1        EQU         $64                       ;AUTO VECTOR #1-DATA COM CONTROL
                                                                  ;This is the VIA used in line
                                                                  ;protocols
00000068        VEC2        EQU         $68                       ;AUTO VECTOR #2-DC 1
00000070        VEC4        EQU         $70                       ;AUTO VECTOR #4-DC 0
                ;
                ;
                ;*****************************************************************************
                ;
                ; Unit I/O Command codes  --found IN D4.W
                ;
00000000        INSTCMD     EQU         0                         ; Install the unit
00000001        READCMD     EQU         1                         ; read command
00000002        WRCMD       EQU         2                         ; write command
00000003        CLRCMD      EQU         3                         ; CLEAR THE UNIT
```

```
          00000004          BUSYCMD     EQU     4                       ; busy command
          00000005          STBCMD      EQU     5                       ; STATUS COMMAND -ACTUAL COMMANDS IN D2.V
          00000006          UNMCMD      EQU     6                       ; unmount command
                            ;
                            ; mode flags -- found in D4.V
                            ;
          0000000C          LFsprsflg   EQU     $0C                     ; Auto LF suppress bits 2 or 3
                            ;
                            ;.   STATUS COMMANDS FOUND IN D2.V

                            , functions compatible with old printer driver

          00000000          D2_FREEW    EQU     0                       ;RETURN WRITE BUFFER FREE SPACE
          00000001          D2_BAUDS    EQU     D2_FREEW+1              ;SET READ/WRITE BAUD RATE
          00000002          D2_PARTY    EQU     D2_BAUDS+1              ;SET PARITY
          00000003          D2_FREER    EQU     D2_PARTY+1              ;RETURN READ BUFFER FREE SPACE
          00000004          D2_CHARS    EQU     D2_FREER+1              ;SET CHARACTER SIZE
          00000005          D2_HANDS    EQU     D2_CHARS+1              ;SET HANDSHAKE METHOD
          00000006          D2_BFCTRL   EQU     D2_HANDS+1              ;RETURN STATE OF BUFFER CONTROL TABLE

                            ; new functions

          00000007          D2_RESTS    EQU     D2_BFCTRL+1             ;RETURN READ STATUS
          00000008          D2_WRSTS    EQU     D2_RESTS+1              ;RETURN WRITE STATUS
          00000009          D2_REAHI    EQU     D2_WRSTS+1              ,SET READ HI WATER MARK(NMBR OF CHARS LEFT F
REE)
          0000000A          D2_REALO    EQU     D2_REAHI+1              ;SET READ LO WATER MARK(NMBR OF CHARACTERS L
FT IN BFR)
          0000000B          D2_OUTRD    EQU     D2_REALO+1              ;USER DISABLE OF OUTBOUND READ (BUFFER DISAB
LE)
          0000000C          D2_INBRD    EQU     D2_OUTRD+1              ,USER DISABLE OF INBOUND READ  (DEVICE DISAB
LE)
          0000000D          D2_OUTWT    EQU     D2_INBRD+1              ;USER DISABLE OF OUTBOUND WRITE (DEVICE DISA
BLE)
          0000000E          D2_INBWT    EQU     D2_OUTWT+1              ;USER DISABLE OF INBOUND WRITE (BUFFER DISAB
LE)
          0000000F          D2_WBCHR    EQU     D2_INBWT+1              ;RETURN THE NUMBER OF CHARACTERS IN WRITE BU
FFER
          00000010          D2_RBCHR    EQU     D2_WBCHR+1              ;RETURN THE NUMBER OF CHARACTERS IN READ BUF
FER
          00000011          D2_ATLF     EQU     D2_RBCHR+1              ;TOGGLE auto LineFeed flag
          00000012          D2_BENQ     EQU     D2_ATLF+1               ;SET number of chars between ENQ's or ETX's
          00000013          D2_RDALTB   EQU     D2_BENQ+1               ;Set Read Alternate Buffer
          00000014          D2_WTALTB   EQU     D2_RDALTB+1             ,Set Write Alternate Buffer
          00000014          TELSTATE    EQU     D2_WTALTB               ;The last status function code
                            ;
                            ;
                            ;           THE ABOVE IS WILD AND WOOLY AND MAY BE OF LITTLE US TO A SIMPLE
                            ;           HIGHER LEVEL PROTOCOL-- HOWEVER THE HIGHER YOU GET THE MORE USE
                            ;           SOME OF THESE REPORTING FUNCTIONS MAY BE
                            ;
                            ;
                            ;
                            ;********************************************************************************
```

```
                              ;
                              ; 68000 status Register values
                              ;
                              ;
         0000A000             UPRMSK     EQU      $A000                 ; KEEPS ALL STATES AND TRACE BITS AS IS
         00002000             STATEMSK   EQU      $2000                 ; ANDS OFF STATE BIT
         00008000             TRACEMSK   EQU      $8000                 ; ANNDS OFF TRACE BIT
         00000700             INTMSK     EQU      $0700                 ; ANDS OFF ALL INT LEVELS
         00000400             INT4       EQU      $400                  ; INTERRUPT LEVEL 4 AND LOWER
         00000200             INT2       EQU      $200                  ; ETC LEVEL2
         00000100             INT1       EQU      $100                  ;ETC LEVEL 1
         00002400             DISINT4    EQU      $2400                 ; Disable all DataCom 0 (priority 4) and be
low device ints
         00002200             DISINT2    EQU      $2200                 ; DISABLE DATACOM 1 (priority 2 )and below
ints
         00002100             DISINT1    EQU      $2100                 ; Disable DataCom-Control int
         00000001             CARRYST    EQU      $0001                 ; CCR with carry set
                              ;
                              ; VIA Addresses
                              ;
         00030F63             ORA        EQU      $30F63                ; PORT A
         00030F67             DDRA       EQU      $30F67                ; PORT A DATA DIRECTION REG.
         00030F7F             NHIRA      EQU      $30F7F                ; PORT A W/O HANDSHAKE(IGNORE DDRA)
                              ;
                              ; VIA register values
                              ;
         00000080             IODDRA     EQU      $80                   ; PORT A BIT CONFIGURATION
                              ;
                              ;*****************************************************************************
                              ;
                              ; UART register definitions
                              ;
         00030F20             UARTDC0    EQU      $30F20                ;BASE ADDRESS OF DATACOM 0 UART
         00000020             DC1OFF     EQU      $20                   ;OFFSET FROM DC0 BASE TO DC1 BASE
         00000001             DATAREG    EQU       1                    ;DATA PORT REGISTER INDEX
         00000003             STATRI     EQU       3                    ;STATUS REGISTER INDEX
         00000005             CMDREGI    EQU       5                      ;COMMAND REGISTER INDEX
         00000007             CTLREGI    EQU       7                      ;CONTROL REGISTER INDEX
                              ;
                              ;   UART STATUS REGISTER EQUATES
                              ;
         00000000             S_PARI     EQU      BITD0                ;PARITY ERROR IF SET--SELF CLEARING
         00000001             S_FRAME    EQU      BITD1                ;FRAMING ERROR IF SET --SELF CLEARING
         00000002             S_OVRN     EQU      BITD2                ;DATA OVERRUN IF SET
         00000003             S_RCVF     EQU      BITD3                ;RECEIVE REGISTER FULL IF SET -CLEARED BY RE
AD DATA
         00000004             S_WRTE     EQU      BITD4                ;WRITE REGISTER EMPTY IF SET
         00000005             S_DCD      EQU      BITD5                ;DATA CARRY DETECT IF LO ---WIRED LOW
         00000006             S_DSR      EQU      BITD6                ;DATA SET READY IF LOW --- WIRED LOW
         00000007             S_IRQ      EQU      BITD7                ;INTERRUPT REQUEST IF SET
                              ;                                 S_RCVF EQUIVALENT TO RCVBF
                              ;                                 S_WRTE EQUIVALENT TO XMITBE
                              ;
                              ; UART status registe masks
                              ;
         00000007             S_ERRBits  EQU      $07                  ;Parity, Framing, and Overrun
                              ;
```

```
                         ;   UART COMMAND REGISTER
                         ;                      NOTE:cannot or members of same section together
                         ;
          00000000       CM_DISP     EQU      0                         ;DISABLE PARITY
          00000020       CM_OPBT     EQU      $20                       ;ODD PARITY BOTH XMIT AND RCV
          00000060       CM_EPBT     EQU      $60                       ;EVEN PARITY BOTH XMIT AND RECEIVE
          000000A0       CM_MPBD     EQU      $A0                       ;MARK PARITY BIT UPON XMIT -PARITY CK DISABL
ED
          000000E0       CM_SPBD     EQU      $E0                       ;SPACE PARITY BIT ON XMIT - PARITY CK DISABL
ED
                         ;--------------------------------------
          00000010       CM_ECHO     EQU      $10                       ;IF SET-ECHO MODE FOR RECEIVER
                         ;--------------------------------------
          00000001       CM_DTRL     EQU      $1                        ;ENABLE RCVR/XMITER IF SET DTR BAR=LOW
                         ;--------------------------------------
          00000002       CM_IRQD     EQU      $2                        ;DISABLE INTERRUPTS IF SET --- NOTE CORVUS C
UTEY
                                                                    ;  THIS IS ENABLED FROM STATUS BIT 3, NOT BI
T 0
                                                                    ;  AS IS INDICATED IN SYNERTEK LITERATURE
                         ;--------------------------------------
          00000000       CM_TDHI     EQU      0                         ;XMIT DISABLED RTS BAR HI
          00000004       CM_TELO     EQU      $4                        ;XMIT ENABLED RTS BAR LO
          00000008       CM_TDLO     EQU      $8                        ;XMIT DISABLED RTS BAR LO
          0000000C       CM_TDBRK    EQU      $C                        ;XMIT DISABLED --XMIT BREAK
                         ;
                         ;                    SOME USEFUL MACRO COMMANDS
                         ;                    FOR THE COMMAND REGISTER
          00000002       TURNOFF     EQU      CM_IRQD
          00000004       XMITENB     EQU      CM_TELO
          00000008       XMITDIS     EQU      CM_TDLO
          00000009       CMDRC       EQU      CM_DTRL+CM_TDLO         ;NO XMIT INT, RCV INT,ENAB DTR, NO PARITY
          00000005       CMDRWC      EQU      CM_DTRL+CM_TELO         ;SAME AS CMDRC XCEPT XMIT INTERRUPTS ENABLED
 ALSO
                         ;
                         ;--------------------------------------
                         ;
          000000F3       CLRD3D2     EQU      $F3                        ;CLEAR BITS D3 & D2 A MASK
                         ;
                         ;*************************************************************
                         ;
                         ;    UART CONTROL REGISTER EQUATES
                         ;
                         ;           NOTE: Baud is lower 4 bits of control word--see BAUDCNV table below
                         ;
          00000080       CR_STPB     EQU      $80                        ;IF 0 THEN = 1 STOP BIT
                                                                     ;  IF SET AS INDICATED = 2 STOP BITS IF NO P
ARITY
                                                                     ;                   =1 STOP BIT IF 8 BIT
CHAR + PARITY
                                                                     ;                   =1.5 STOP BITS IF 5BI
T WORD NO PARITY
                         ;--------------------------------------
          00000000       CR_WRDL8    EQU      0                          ;8 BITS WORD LENGTH
          00000020       CR_WRDL7    EQU      $20                        ;7 BIT WORD LENGTH
          00000040       CR_WRDL6    EQU      $40                        ; 6 ETC
          00000060       CR_WRDL5    EQU      $60                        ; 5 ETC.
                         ;--------------------------------------
          00000000       CR_EXTCLK   EQU      0                          ;EXTERNAL RECEIVE CLOCK
          00000010       CR_BRCLK    EQU      $10                        ;BAUD RATE GEN FOR CLOCK
```

```
                       , UART CONTROL REGISTER CONSTANTS FOR UART SETUP
                       ,
         00000016      CTLRC      EQU      CR_BDCLK+CR_WRDL8        ,1 STOP BIT,8BIT WORD LENGTH,BAUD RATE GENE
RATOR

                       , ASCII Control characters for printer control
                       ,
         00000011      XON        EQU      $11                      ,CAN XMIT (CTL-Q)
         00000013      XOFF       EQU      $13                      ,STOP XMIT (CTL-S)
         00000003      ETX        EQU      $03                      ,READY FOR MORE? (CTL-C)
         00000005      ENQ        EQU      $05                      ,READY FOR MORE? (CTL-E)
         00000006      ACK        EQU      $06                      ,YES, I'M READY (CTL-F)
         00000000      NULL       EQU      $00                      ,NULL CHARACTER-DO NOTHING
         0000000D      CR         EQU      $0D                      ,CARRIAGE RETURN
         0000000A      LF         EQU      $0A                      ,LINE FEED
                       ,
                       , Maximum Parameter values for Unitstatus Set table entry functions
                       ,
         00000006      MAXBAUD    EQU      6                        ,FOR SET BAUD RATE
         00000004      MAXPRTY    EQU      4                        ,FOR SET PARITY
         00000001      MAXWRDS    EQU      1                        ,FOR SET WORD SIZE
         00000001      MAXDTCM    EQU      1                        ,FOR SET DATACOM
         00000009      MAXHNDS    EQU      9                        ,FOR SET HANDSHAKE TYPE
         00000085      MAXWHI     EQU      133                      ,HI WATER WRITE MAX # CHARS
         00000050      MAXWLO     EQU      80                       ;LO WATER WRITE MAX # CHARS
         00000085      MAXRHI     EQU      133                      ,HI WATER READ #CHARS MAX
         00000050      MAXRLO     EQU      80                       ,LO WATER READ #CHARS MAX
                       ,
                       , error codes (IORESULT)
                       ,
         00000003      INVCMD     EQU      IOEioreq                 ,invalid cmd-(invalid I/O request)
         00000032      INVTBLID   EQU      IOEtblid                 ,invalid table id
         00000036      INVPRM     EQU      IOEuiopm                 ,invalid parameter
         00000038      INVFNC     EQU      IOEfnccd                 ,invalid function code
                       ;
                       , Miscellaneous definitions
                       ,
         00000001      TRUE       EQU      1                        ; Pascal true boolean value
         00000001      ON         EQU      1                        ,LISTING CONTROL - START LISTING
         00000000      OFF        EQU      0                        ,LISTING CONTROL - STOP LISTING
         000000F0      HILOMSK    EQU      $F0                      ,MASK OFF WATER MARKS -THRO THEM AWAY
```

```
;
; UNIT I/O PARAMETER PASSING DEFINITION
;
;   COMMAND       UNIT   ADDR   COUNT   BLOCK   MODE    IORESULT   BUSY
; 0 - INSTALL     D0.W                                  D7.W
; 1 - READ        D0.W   D1.L   D2.W                    D7.W
; 2 - WRITE       D0.W   D1.L   D2.W                    D7.W
; 3 - CLEAR       D0.W                                  D7.W
; 4 - BUSY        D0.W                                  D7.W       D0.W
; 5 - STATUS      D0.W   D1.L   D2.W <--FUNCTION CODE   D7.W
; 6 - UNMOUNT     D0.W                                  D7.W
;
; ALL REGISTER VALUES ON ENTRY ARE SAVED AND RESTORED EXCEPT D0 & D7
; INTERNAL REGISTER USEAGE :
;
;       D0      =       temp reg
;       D1      =       temp reg
;       D2      =       user's count
;       D3      =       character to or from buffer
;       D4      =       unit number
;       D5      =       Mode flag & in DC ctl int rtns - VIA line bit 0
;       D6      =       save of SR
;       A0      =       temp reg
;       A1      =       temp reg
;       A2      =       temp reg
;       A3      =       Parameter block address (user's data)
;       A4      =       buffer address in SetupWB and SetupRB
;       A5      =       UART base address
;       A6      =       Pointer to port's data area
;
```

```
                                  GLOBAL    COMDRV
                        ;
                        ; DATACOM DRIVER
                        ;
0000                    COMDRV
0000  601E                        BRA.S     COM001                    ;JUMP AROUND HEADER
0002  00                          DATA.B    0                         ;DEVICE NOT BLOCKED
0003  1F                          DATA.B    31                        ;VALID CMDS - ALL VALID
0004  53 01 0E 00                 DATA.B    83,01,14,00               ;DATE
0008  17                          DATA.B    hmlen                     ;HEADER MSG LENGTH
0009  44415441434F4D4D  xxx010    DATA.B    'DATACOMM driver (v 5.0)' ;HEADER MSG
0011  2064726976657220
0019  287620352E3029
      00000017          hmlen     EQU       %-xxx010
                        ;
0020  7E03              COM001    MOVEQ     #INVCMD,D7                ;assume invalid command
0022  0C44  0006                  CMPI.W    #NUMCMD,D4                ;VALID COMMAND?
0026  622C                        BHI.S     PRNDERR                   ;NO
0028  48E7  7EFE                  MOVEM.L   D1-D6/A0-A6,-(SP)         ;SAVE REGISTERS
002C  4287                        CLR.L     D7                        ;CLEAR IORESULT
002E  2641                        MOVEA.L   D1,A3                     ;ADDRESS OF USERS BUFFER
0030  4286                        CLR.L     D6                        ;Clear save of SR register
0032  4DFA  09FA+                 LEA       PORT0Data, A6             ; assume talking to Port 0
0036  B07A  0E80+                 CMP.W     UnitP0, D0                ;is it Port 0?
003A  6704                        BEQ.S     COMisP0                   ;yes
003C  4DFA  0C4C+                 LEA       PORT1Data, A6             ;NO, talking to port 1
                        ;
0040  C144              COMisP0   EXG       D0, D4                    ;save unit number
0042  43FA  0012+                 LEA       COMTBL,A1                 ;TURN THE COMMAND INTO A
0046  E348                        LSL.W     #1,D0                     ;INDEX TO THE FUNCTION
0048  3031  0000                  MOVE.W    0(A1,D0.W),D0
004C  4EB1  0000                  JSR       0(A1,D0.W)                ;DO FUNCTION
0050  4CDF  7F7E                  MOVEM.L   (SP)+,D1-D6/A0-A6         ;Restore registers
0054  4E75              PRNDERR   RTS
                        ;
                        ; THE PRINTER DRIVER JUMP TABLE
                        ;
0056  000E              COMTBL    DATA.W    COMINST-COMTBL            ;UNITINSTALL
0058  0164                        DATA.W    COMRD-COMTBL              ;UNITREAD
005A  0260                        DATA.W    COMWR-COMTBL              ;UNITWRITE
005C  05FE                        DATA.W    COMCLR-COMTBL             ;UNITCLEAR
005E  066E                        DATA.W    COMBSY-COMTBL             ;UNITBUSY
0060  06B0                        DATA.W    COMST-COMTBL              ;UNITSTATUS
0062  067C                        DATA.W    COMUNMT-COMTBL            ;UNITUNMOUNT
```

```
                        ;
                        ; COMINST - UNITINSTALL ==> SETUP THE DEFAULT BUFFER CONTROL FEATURES
                        ;   Assumes that a spurious DataCom Control interrupt is benign and will
                        ;    be handled by the DataCom Control interrupt service routine correctly.
                        ;
                        ; save unit number and toggle common flag
                        ;
0064 6160               COMINST   BSR.S    SaveUnit
0066 612E                         BSR.S    DISINTS               ;DISABLE DATACOM INTERRUPTS

                        ; init buffer control table
                        ;
0068 41EE 000A                    LEA      BFRCTL(A6), A0        ;beginning of table
006C 43EE 0000                    LEA      DEFBWRT(A6), A1       ;beginning of default table
0070 7004                         MOVEQ    #DEFBCTLN-1, D0       ;number of words in table

0072 30D9               CINbufctl MOVE.W   (A1)+, (A0)+          ;move from default to real
0074 51C8 FFFC                    DBF      D0, CINbufctl         ;table is even number of words
                        ;
                        ; Initialise UART from constants and Printer Control Table & initialise VIA
                        ;
0078 13FC 0000 0003               MOVE.B   #IODDRA,DDRA.L        ;INITIALIZE DATA DIRECTION REG FOR PORT A
007E 0F47
0080 6162                         BSR.S    SETUART
                        ;
                        ; Initialise READ, WRITE AND CONTROL BUFFER CONTROL TABLES
                        ;
0082 6100 00CE                    BSR      INIWRBF               ;init write buffer
0086 6100 00EE                    BSR      INIRDBF               ;init read buffer
008A 6100 0118                    BSR      INITCTLB              ;init control buffer
                        ;
                        ; Setup interrupt vectors
                        ;
008E 6100 0092                    BSR      SETVECS
                        ;
                        ; If saved SR then restore it
                        ;
0092 6128                         BSR.S    ENBINTS
0094 4E75                         RTS
```

```
                        ;
                        ; DISINTS - disable interrupts for Port selected.  If Port 0 then disable up to
                        ;           level 4.  If Port 1 selected than disable up to level 2.
                        ;
                        ;       Entry : D6 = saved SR if not zero
                        ;               D4 = unit number
                        ;       Exit  : D6 = saved SR or zero
                        ;
0096  323C  0400        DISINTS     MOVE.W    #INT4, D1             ;assume Port 0, level 4 int
009A  B87A  0E4C+                   CMP.W     UnitP0, D4            ;is it Port 0?
009E  6704                          BEQ.S     DITisP0               ;yes
00A0  323C  0200                    MOVE.W    #INT2, D1             ;no, use Port 1 level 2 int
00A4  40C0              DITisP0     MOVE.W    SR,D0                 ;get current status register
00A6  0240  0700                    ANDI.W    #INTMSK,D0            ;GET ONLY INTERRUPT LEVELS
00AA  B041                          CMP.W     D1, D0                ;is current < current Port's level
00AC  640C                          BCC.S     DITexit               ;no, exit
                        ;
                        ;    NOW set up disable with minimum disturbance of upper level
                        ;            status bits --- this too wont work if user and
                        ;            supervisor space are both utilised.
                        ;
00AE  40C6                          MOVE.W    SR, D6                ;save current SR
00B0  40C0                          MOVE.W    SR, D0                ;get current status register for change
00B2  0240  A000                    ANDI.W    #UPRMSK,D0            ;KEEP ONLY UPPER BITS
00B6  8041                          OR.W      D1, D0                ;disable current Port's level
00B8  46C0                          MOVE.W    D0,SR                 ;turn off the ints in the SR
00BA  4E75              DITEXIT     RTS
                        ;
                        ; ENBINTS - Restore saved SR if saved it
                        ;       Entry : D6 = saved SR if not zero
                        ;       Exit  : D6 = if D6 was not zero then SR <- D6 and D6 <- 0
                        ;                    otherwise SR remains untouched and D6 stays 0
                        ;
00BC  4A46              ENBINTS     TST.W     D6                    ;Does D6 have a saved SR
00BE  6704                          BEQ.S     EITEXIT               ;DIDN'T SAVE SO EXIT
00C0  46C6                          MOVE.W    D6,SR                 ;restore SR
00C2  4286                          CLR.L     D6                    ;always leave D6 = to zero
00C4  4E75              EITEXIT     RTS
```

```
                     ;
                     ; SaveUnit - determine if this is Port 0 or Port 1 and save unit number
                     ;           also initialize A6 to address of port's data area
                     ;
                     ;      Entry : D4 = unit number
                     ;      Exit  : A6 = address of port's data area
                     ;
00C6  43FA  0E20+      SaveUnit   LEA     UnitP0, A1          ;assume is Port 0
00CA  4DFA  0962+                 LEA     Port0Data, A6

00CE  41FA  0E16+                 LEA     CMNFLGS, A0         ;if portflg flag was
00D2  0850  0000                  BCHG    #PORTFLG, (A0)      ; zero then is port 0
00D6  6708                        BOFF.S  SVUisP0             ;else it is now port 1

00D8  43FA  0E10+                 LEA     UnitP1, A1          ;Port 1 addresses
00DC  4DFA  0BAC+                 LEA     Port1Data, A6

00E0  3284             SVUisP0    MOVE.W  D4, (A1)            ;save unit number
00E2  4E75                        RTS
```

```
                        ;
                        ; SETUART - Initialize UART from constants and Buffer Control Table
                        ;
                        , Get UART Register Base address
                        ,
00E4 612A               SETUART    BSR.S      GETBASE                  ;RETURNS BASE IN A0
                        ,
                        , Setup UART's Control register - index = 7  from Base
                        ,
00E6 7010                          MOVEQ      #CTLRC,D0                ,1 STOP BIT,BAUD RATE GEN
00E8 122E 000D                     MOVE.B     BF_WRDS(A6), D1          .ADD WORD SIZE-7 OR 8 BITS
00EC EB09                          LSL.B      #5,D1                    ,MOVE INTO HI ORDER BITS
00EE 8001                          OR.B       D1,D0                    ;00=8 BITS,01=7 BITS
00F0 802E 000B                     OR.B       BF_RDBD(A6), D0          ;ADD BAUD RATE FROM TABLE
00F4 1B40 0007                     MOVE.B     D0,CTLREGI(A5)           .PUT IN CONTROL REGISTER
                        ,
                        , Setup UART's Command register - index = 5 from Base
                        , make transmit buffer empty interrupt enabled - when occurs int rtn will
                        ,  turn off if buffers are empty.
                        ;
00F8 7005                          MOVEQ      #CMDRWC, D0              ;CMD CONSTANTS xmit int enabled
00FA 122E 000C                     MOVE.B     BF_PART(A6), D1          ;GET TABLE PARITY
00FE EB09                          LSL.B      #5,D1                    ,PUT IN CORRECT BIT POSITION
0100 8001                          OR.B       D1,D0
0102 1B40 0005                     MOVE.B     D0,CMDREGI(A5)           ,PUT IN COMMAND REGISTER
                        ,
                        , Read the Data Port and Status Register to clear all Status flags
                        ,
0106 102D 0001                     MOVE.B     DATAREG(A5),D0           ,DATA PORT AT INDEX = 1
010A 102D 0003                     MOVE.B     STATRI(A5),D0            ;STATUS REG AT INDEX = 3
010E 4E75                          RTS
```

```
                                ;
                                , GETBASE - Get address of UART's register Base address in memory
                                ,              Entry . D4 = unit number
                                ,              EXIT . (A5) = Base address
                                ,
0110  4BF9  0003  0F20    GETBASE      LEA        UARTDC0.L, A5            ,ASSUME USING DATACOM 0
0116  B87A  0DD0+                      CMP.W      UnitP0, D4               ,is it Port 0?
011A  6704                             BEQ.S      GBSisP0                  ,yes
011C  DAFC  0020                       ADDA.W     #DC1OFF, A5              ,No, BASE  = OFFSET+UART DC0 BASE ADDR
0120  4E75                GBSisP0      RTS
                                ,
                                , SETVECS - Put interrupt routine's entry addresses into the interrupt vectors
                                ,              If Port 0 put in DC 0 int rtn address in Vector 4
                                ,              otherwise assume is port 1
                                ,              Saves old level 1 interrupt vector if it is not = to this driver's
                                ,                 interrupt routines address.
                                ,
                                ,              Entry   D4 = unit number
                                ,                      interrupts disabled to level for current Port
                                ,
0122  41FA  049C+         SETVECS      LEA        DCTLINT,A0               ,PUT DATA COM CONTROL
0126  2278  0064                       MOVEA.L    VEC1.W, A1               ,Get old vector
012A  21C8  0064                       MOVE.L     A0,VEC1.W                .INT ROUTINE IN VEC 1
012E  B3C8                             CMPA.L     A0, A1                   .should save old vector
0130  6706                             BEQ.S      SVCsame                  ,no, they're the same
0132  41FA  0DB0+                      LEA        SaveLvl1, A0             .yes save in common area
0136  2089                             MOVE.L     A1, (A0)

0138  B87A  0DAE+         SVCsame      CMP.W      UnitP0, D4               ,is it Port 0?
013C  660A                             BNE.S      SVCdoP1                  ,no, do level 2 for Port 1

013E  41FA  025C+                      LEA        DC0INT,A0                ,ADDR OF DC0 entry point to XMIT/RCV INT RO
UTINE
0142  21C8  0070                       MOVE.L     A0,VEC4.W                ,put it in VEC 4
0146  6008                             BRA.S      SVCexit

0148  41FA  0260+         SVCdoP1      LEA        DC1INT,A0                ,ADDR OF DC1 entry point to XMIT/RCV INT RO
UTINE
014C  21C8  0068                       MOVE.L     A0,VEC2.W                ,put it in VEC 2

0150  4E75                SVCexit      RTS
```

```
                           ;
                           ; INIWRBF - Initialise Write Buffer variables to EMPTY Buffer also ENQ, BUSY and
                           ;           SENDLF are cleared to false.  Use default buffer
                           ;
0152  41EE  0014           INIWRBF   LEA      WRTCTL(A6),A0           ;WRITE BUFFER CONTROL TABLE
0156  4258                           CLR.W    (A0)+                   ;RESET ALL FLAG 1
0158  4218                           CLR.B    (A0)+                   ;RESET ALL FLAG 2 except
015A  08D0  0004                     BSET     #AULF_W2,(A0)           ; DO AUTO LINE FEED and *kb 1/5/83*
015E  08D8  0001                     BSET     #EMPT_W2,(A0)+          ; BUFFER IS EMPTY
0162  43EE  015C                     LEA      WRTBUF(A6), A1          ;WRITE BUFFER
0166  20C9                           MOVE.L   A1,(A0)+                ;FILL POINTER (USED TO FILL CHARACTERS IN)
0168  20C9                           MOVE.L   A1,(A0)+                ;EMPTY POINTER (USED TO EMPTY CHARACTERS OU
T)
016A  20C9                           MOVE.L   A1,(A0)+                ;Save buffer address
016C  30FC  0100                     MOVE.W   #WBFLEN,(A0)+           ;MAXIMUM SIZE OF BUFFER
0170  30FC  0100                     MOVE.W   #WBFLEN,(A0)+           ;NUMBER OF LOCATIONS AVAILABLE TO FILL
0174  4E75                           RTS
                           ;
                           ; INIRDBF - Initialise READ Buffer variables to EMPTY Buffer also ENQ, BUSY and
                           ;           SENDLF are cleared to false.  Use default buffer
                           ;
0176  41EE  0030           INIRDBF   LEA      RDCTL(A6), A0           ;READ BUFFER CONTROL TABLE
017A  4258                           CLR.W    (A0)+                   ;RESET ALL FLAG 1
017C  4218                           CLR.B    (A0)+                   ;RESET ALL FLAG 2 except,
017E  08D8  0001                     BSET     #EMPT_W2,(A0)+          ;   BUFFER IS EMPTY
0182  43EE  025C                     LEA      RDBUF(A6),A1            ;READ BUFFER
0186  20C9                           MOVE.L   A1,(A0)+                ;FILL POINTER (USED TO FILL CHARACTERS IN)
0188  20C9                           MOVE.L   A1,(A0)+                ;EMPTY POINTER (USED TO EMPTY CHARACTERS OU
T)
018A  20C9                           MOVE.L   A1,(A0)+                ;Save buffer address
018C  30FC  0100                     MOVE.W   #RBFLEN,(A0)+           ;MAXIMUM SIZE OF BUFFER
0190  30FC  0100                     MOVE.W   #RBFLEN,(A0)+           ;NUMBER OF LOCATIONS AVAILABLE TO FILL
0194  4298                           CLR.L    (A0)+                   ;Clear alternate buffer address
0196  4258                           CLR.W    (A0)+                   ;Clear alternate buffer length
0198  30FC  0085                     MOVE.W   #MAXRHI,(A0)+           ;NUMBER OF CHARACTERS FOR HIGH WATER MARK
019C  30FC  0050                     MOVE.W   #MAXRLO,(A0)+           ;NUMBER OF CHARACTERS FOR LOW WATER MARK
01A0  4258                           CLR.W    (A0)+                   ;CLEAR ENQ COUNT
01A2  4E75                           RTS
                           ;
                           ; INITCTLB  - Initialize the control character buffer to empty
                           ;
01A4  41EE  0058           INITCTLB  LEA      CTLBUF(A6), A0           ;CONTROL CHARACTER BUFFER
01A8  43EE  004E                     LEA      CB_FRONT(A6), A1         ;CTL CHAR BUF TABLE ADDRESS
01AC  22C8                           MOVE.L   A0, (A1)+                ;set front and
01AE  22C8                           MOVE.L   A0, (A1)+                ;rear pointers to begin of buffer
01B0  4251                           CLR.W    (A1)                     ;clear all flags except
01B2  08E9  0001  0001               BSET     #EMPT_CB, 1(A1)          ;buffer empty
01B6  4E75                           RTS
```

```
                         , COMRD - UNITREAD   READ FROM THE DATACOM BUFFER
                         ,
                         ,           INPUTS...  .D2 COUNT OF CHARACTERS THE USER WANTS TO READ
                         ;                       D4 unit number
                         ,                       A3 ADDRESS OF USER'S BUFFER
                         ,                       A6 Address of ports data area
                         ,
                         ,  NOTES  For reading, interrupts will occur when the input buffer is full -no
                         ,         priming is necessary as is with writing. Also if full duplex activities
                         ,         then a read and write interrupt may be the same interrupt -have to check
                         ,         status flags of UART
                         ; First see if user's count is exhausted if not attempt a read
                         ,
01BA  4A42               COMRD       TST.W     D2
01BC  6736                           BEQ.S     COMREX                    ,COMREX GENERAL EXIT ROUTINE

                         , Check if the user has disabled output - Buffer to User
                         ,
01BE  082E  0004  0031   REREAD      BTST      #OUTE_R1,RB_FLG1+1(A6)    ,IS BUFFER TO USER TRANSFER ENABLED?
01C4  6704                           BEQ.S     CKRdErr                   , YES, check for input error on UART
01C6  7E3D                           MOVEQ     #IOEordsbl,D7             ,no, tell user can't => ERROR
01C8  602A                           BRA.S     COMREX

                         , Check for a UART error
                         ;
01CA  08AE  0001  0031   CKRdErr     BCLR      #Err_R1, RB_FLG1+1(A6)    ,Have a read error
01D0  6704                           BOFF.S    CKPORT                    ,no, see if have data
01D2  7E43                           MOVEQ     #IOEuarter,D7             ,yes, tell user and exit
01D4  601E                           BRA.S     COMREX

                         , if there is any data in the buffer, give it to user. If there is no data and
                         ,  the user has disabled the inbound read, remind him. However if there is no data
                         ;  put him in a loop waiting for data.
                         ,
01D6  082E  0001  0033   CKPORT      BTST      #EMPT_R2,RB_FLG2+1(A6)    ,BUFFER IS EMPTY?
01DC  670C                           BOFF.S    READONE                   , NO, GO READ A CHARACTER
01DE  082E  0005  0031               BTST      #INPE_R1,RB_FLG1+1(A6)    , yes, INPUT ENABLED?
01E4  67D8                           BOFF.S    REREAD                    , YES, wait for a char
01E6  7E3C                           MOVEQ     #IOEirdsbl,D7             ,no, tell user input is disabled
01E8  600A                           BRA.S     COMREX

                         , get user his characters and manage buffer
                         ,
01EA  610A               READONE     BSR.S     UGETCHR                   ,GET THE CHARACTER FOR THE USER FROM THE BUF
FER
01EC  6506                           BCS.S     COMREX                    ,exit if error, D7 has error code

                         ; Put character in user's buffer and update
                         ;
01EE  16C3                           MOVE.B    D3,(A3)+                  ,update buffer pointer
01F0  5342                           SUBQ.W    #1,D2                     ,subtract one from user count
01F2  60C6                           BRA.S     COMRD                     ;GETSMOA IF AVAILABLE
                         ,
01F4  4E75               COMREX      RTS
```

```
                 ; UGETCHR  --- User level get character routine, gets the character from the read buffer.
                 ;
                 ;       Entry     A6  = pointer to ports data area
                 ;                 D2  = user count
                 ;                 D4  = unit number
                 ;                 buffer is NOT empty
                 ;       Exit      D3  = character if one gotten
                 ;                 (C) = Error, D7 has error code
                 ;                (NC) = got a character no error
                 ;
01F6 6100 FE9E     UGETCHR    BSR        DISINTS                  ;disable interupts

01FA 206E 0038                MOVE.L     RB_EMPTY(A6), A0         ;A0 =/ EMPTYING POSITION OF RD BUFFER
01FE 1618                     MOVE.B     (A0)+,D3                 ;Get chars
0200 2D48 0038                MOVE.L     A0, RB_EMPTY(A6)         ;Save the new Front pointer in rb_empty
                 ;
                 ; Update buffer variables
                 ;
0204 226E 003C                MOVE.L     RB_BADR(A6), A1          ;A1 = ADDRESS OF BUFFER BEGIN
0208 D2EE 0040                ADDA.W     RB_SIZE(A6), A1          ;A1 = ADDRESS OF END OF BUFFER
020C B1C9                     CMPA.L     A1, A0                   ;Is Front pointing beyond buffer?
020E 6306                     BLS.S      UGCnowrp                 ;No, don't do wrap around
                 ;                                                ;Yes, set front = addr 1st byte of buffer
0210 2D6E 003C 0038           MOVE.L     RB_BADR(A6), RB_EMPTY(A6) ;Save the new Front pointer in rb_empty
                 ;
0216 526E 0042     UGCnowrp   ADDQ.W     #1, RB_FREE(A6)          ;SINCE WE GOT CHAR, ONE MORE FREE SPACE
                 ;
                 ; see if buffer is empty
                 ;
021A 322E 0042                MOVE.W     RB_FREE(A6), D1
021E B26E 0040                CMP.W      RB_SIZE(A6), D1          ; # OF FREE LOCATIONS - BUFFER SIZE
0222 650A                     BCS.S      UGCnotmt                 ; not empty if free < size
0224 6212                     BHI.S      HELPRD                   ; HELPHD IS SERIOUS ERROR (free > size)
                 ;
0226 08EE 0001 0033           BSET       #EMPT_R2, RB_FLG2+1(A6)  ; BUFFER empty
022C 615C                     BSR.S      RChkAltBf                ; see if should switch to an Alternate buffe

                 ;
                 ; Do protocol control, see if can turn off Read Busy
                 ;
022E 6114          UGCnotmt   BSR.S      ChkProto                 ;check protocol
0230 6100 FE8A                BSR        ENBINTS                  ;enable interrupts
0234 4280                     CLR.L      D0                       ; clear carry
0236 4E75                     RTS
                 ;
0238 6100 FE82     HELPRD     BSR        ENBINTS                  ; SERIOUS BUMMER BUG
023C 7E40                     MOVEQ      #IOEbsserr,D7            ;SIZING ERROR
023E 44FC 0001                MOVE.W     #1,CCR                   ;SET CARRY
0242 4E75                     RTS
```

```
                         ;
                         ;  ChkProto  - checks low water mark for reading to see if should turn off read busy
                         ;  GoUnbusy - entry point to turn off busy state on receives
                         ;
                         ;             NOTES. This routine assumes that interrupts are disabled prior
                         ;                    to its being invoked
0244                     ChkProto
                         ;
                         ; check if input disabled. Cannot turn off busy if is disabled.
                         ;
0244 082E 0005 0031                BTST     #INPE_R1, RB_FLG1+1(A6)
024A 663C                          BON.S    CPRexit                 ;input disabled exit
                         ;
                         ; if (protocols enabled) and (NOT Line type) then check if busy
                         ;
024C 082E 0000 0012                BTST     #PROT_P2, BF_PROF(A6)   ;protocol enabled?
0252 6734                          BOFF.S   CPRexit                 ;no, exit
0254 082E 0000 0013                BTST     #LINE, BF_PROF+1(A6)    ;Line type
025A 662C                          BON.S    CPRexit                 ;yes, exit
                         ;
                         ; if busy then check if buffer at or below low water mark
                         ;
025C 082E 0000 0031                BTST     #BUSY_R1, RB_FLG1+1(A6)
0262 6724                          BOFF.S   CPRexit                 ;not busy exit
                         ;
                         ; if buffer at or below low water mark then turn off busy
                         ;
0264 322E 0040                     MOVE.W   RB_SIZE(A6), D1         ;BUFFER SIZE (ADDRESS OF)
0268 926E 0042                     SUB.W    RB_FREE(A6), D1         ;D1 = number of chars in buffer
026C B26E 004C                     CMP.W    RB_LOWA(A6), D1         ; at or below low water mark?
0270 6216                          BHI.S    CPRexit                 ; No, exit
                         ;
                         ; is at or below when busy so turn off busy
                         ;
0272 08AE 0000 0031      GoUnbusy  BCLR     #BUSY_R1, RB_FLG1+1(A6) ;clear busy state
0278 7011                          MOVEQ    #XON, D0                ;assume XON/XOFF protocol
027A 082E 0001 0013                BTST     #XONXOFF, BF_PROF+1(A6) ;send byte to other side saying not busy
0280 6602                          BON.S    CPRxon                  ;send XON
0282 7006                          MOVEQ    #ACK, D0                ;either ETX/ACK or ENQ/ACK so send ACK
0284 6100 00EC           CPRxon    BSR      PutCtl                  ;send the control char
0288 4E75                CPRexit   RTS
```

```
                    ;
                    ; RChkAltBf - check if alternate buffer switch on read buffer
                    ;        Rcv input is automatically disabled when user calls unitstatus
                    ;        switch buffers.
                    ;
                    ;        Entry    A6 = address of port's datat area
                    ;                         interrupts disabled
                    ;
028A  082E  0001  0033  RChkAltBf   BTST      #EMPT_R2, RB_FLG2+1(A6)  ;is buffer empty?
0290  6722                          BOFF.S    rCABexit                  ;no, can't switch
0292  08AE  0002  0031              BCLR      #ALTBF_R1, RB_FLG1+1(A6) ;is an alternate buffer available?
0298  671A                          BOFF.S    rCABexit                  ;no, nothing to switch
                    ;
                    ; Switch buffers by making the Alternate buffer the main buffer
                    ;
029A  206E  0044                    MOVE.L    RB_ABADR(A6), A0          ;get new buffer address
029E  302E  0048                    MOVE.W    RB_ASIZE(A6), D0          ;and length
02A2  6100  0750                    BSR       SetupRB                   ;switch buffer in table
                    ;
                    ; if user is NOT controlling the input disable bit then enable RCV input
                    ;
02A6  41EE  0031                    LEA       RB_FLG1+1(A6), A0         ;EnbRcvIn needs A0 -> flag byte
02AA  0810  0007                    BTST      #INPC_R1, (A0)            ;is user controlling input disable?
02AE  6604                          BON.S     rCABexit                  ;user is controlling, exit
02B0  6100  0675                    BSR       EnbRcvIn                  ;enable receive input
                    ;
02B4  4E75              rCABexit    RTS
```

```
                                 , COMWR - UNITWRITE
                                 ,
                                 ,           INPUTS  . . D2 COUNT OF CHARACTERS THE USER WANTS TO WRITE
                                 ,                       D4 unit number
                                 ,                       A3 ADDRESS OF USER'S CHARACTERS
                                 ,                       A4 Address of ports data area
                                 ,
                                 , NOTE  For writing, the UART has to be primed to interrupting when the xmit buffer
                                 ,       is empty by enabling the xmit interrupt  If no xmissions then of course its empty
                                 ,       and it interrupts forever  Hence trickery only when sending first of a stream
                                 ,       (starting interrupts) and last of a stream (stopping the little dears)
                                 ,
02B6  4A42                       COMWR       TST.W     D2                       ,IS USER COUNT DONE?
02B8  673E                                   BEQ.S     COMWEX                   ,YES

                                 , if input to buffer disabled input
                                 ,
02BA  082E  0005  0015           REWRITE     BTST      #INPE_W1, WB_FLG1+1(A6)  ,IS USER TO BUFFER TRANSFER ENABLED?(INBOUND
 WRITE)
02C0  6704                                   BOFF.S    CKbuferr                 ,YES, chk if buffer size err found in xmit i
 [illegible]
02C2  7E3E                                   MOVEQ     #IOEiwdsbl,D7            ,input disabled give error
02C4  6032                                   BRA.S     WRPROB                   ,exit

                                 , check write error flag for error during xmit interrupt
                                 ,
02C6  08AE  0001  0015           CKbuferr    BCLR      #ERR_W1, WB_FLG1+1(A6)   ,Error?
02CC  6704                                   BOFF.S    CKWRTP                   ,No, chk if buffer is full
02CE  7E41                                   MOVEQ     #IOEwsserr,D7            ;SIZING ERROR with write buffer
02D0  6026                                   BRA.S     WRPROB                   ,exit

                                 , Check if Buffer is full  if is and output is NOT disabled then spin wheels
                                 ,
02D2  082E  0000  0017           CKWRTP      BTST      #FULL_W2, WB_FLG2+1(A6)  ,Buffer full?
02D8  670C                                   BEQ.S     WRTONE                   ,NO, GO WRITE A CHAR TO THE BUFFER

02DA  082E  0004  0015                       BTST      #OUTE_W1, WB_FLG1+1(A6)  , Yes, OUTPUT IS AT ALL ENABLED?
02E0  67D8                                   BOFF.S    REWRITE                  , YES, spin wheels while buffer emptys
02E2  7E3F                                   MOVEQ     #IOEowdsbl,D7            ,can't send tell user ERROR
02E4  6012                                   BRA.S     WRPROB                   ,exit

                                 , Buffer not full so put user characters or LF into write_buffer
                                 ,
02E6  760A                       WRTONE      MOVEQ     #LF, D3                  ,assume just sent an CR so must send a LF
02E8  08AE  0003  0017                       BCLR      #SNDLF_W2, WB_FLG2+1(A6) ,should send an Line Feed char?
02EE  6604                                   BON.S     WRTanLF                  ,yes
02F0  161B                                   MOVE.B    (A3)+,D3                 ,no, then get 1 user char
02F2  5342                                   SUBQ.W    #1,D2                    ,subtract 1 from user's count
02F4  6104                       WRTanLF     BSR.S     UPUTCHR                  ,PUT THE USER'S CHARACTER INTO THE WRITE BUF
FER
02F6  60BE                                   BRA.S     COMWR
02F8                             COMWEX
02F8  4E75                       WRPROB      RTS
```

```
                          ; UPUTCHR  --- User level put character routine, puts the character into the write buffer.
                          ,
                          ;      Entry : (D3) = character to put in write buffer
                          ;               Buffer is NOT full
                          ;
02FA  6100  FD9A          UPUTCHR     BSR        DISINTS                  ,disable interrupts

02FE  206E  0018                      MOVE.L     WB_FILLP(A6), A0         ,A0 =) FILLING POSITION OF WRITE BUFFER
0302  10C3                            MOVE.B     D3, (A0)+                ;PUT char
0304  2D48  0018                      MOVE.L     A0, WB_FILLP(A6)         ,Save the new Rear pointer in wb_fillp
                          ;
                          , Update buffer variables
                          ;
0308  226E  0020                      MOVE.L     WB_BADR(A6), A1          ,A1 = ADDRESS OF BUFFER BEGIN
030C  D2EE  0024                      ADDA.W     WB_SIZE(A6), A1          ,A1 = ADDRESS OF END OF BUFFER
0310  B1C9                            CMPA.L     A1, A0                   ;Is Rear pointing beyond buffer?
0312  6306                            BLS.B      UPCnowrp                 ;No, don't do wrap around
                          ,                                                ;Yes, set front = addr 1st byte of buffer
0314  2D6E  0020  0018                MOVE.L     WB_BADR(A6), WB_FILLP(A6) ;Save the new Rear pointer in wb_fillp
                          ;
031A  536E  0026          UPCnowrp    SUBQ.W     #1, WB_FREE(A6)          ;SINCE WE TOOK CHAR, ONE LESS FREE SPACE
                          ;
                          ; see if buffer is full  (WB_FREE is an unsigned word)
                          ,
031E  6606                            BNE.S      UPCnotfl                 ,not full, subtract sets or clears ZERO bit
0320  08EE  0000  0017                BSET       #FULL_W2, WB_FLG2+1(A6)  , BUFFER full
                          ;
                          , check if last char is CR.  If is see if should send an LF next time
                          .
0326  0C03  000D          UPCnotfl    CMPI.B     #CR, D3
032A  6616                            BNE.S      UPCnotCR                 ;not a CR
032C  082E  0004  0017                BTST       #AULF_W2, WB_FLG2+1(A6)  ;is it auto LF mode
0332  670E                            BOFF.S     UPCnotCR                 ,no, don't send an LF  *kb 1/5/82*
0334  3005                            MOVE.W     D5, D0                   ,save mode flag  *kb 1/24/83*
0336  0240  000C                      ANDI.W     #LFsprsflg, D0           ,if LF suppress flag set *kb 1/24/83*
033A  6606                            BNE.S      UPCnotCR                 ;then don't send a LF
033C  08EE  0003  0017    UPCisaLF    BSET       #SNDLF_W2, WB_FLG2+1(A6) ,send LF only if D5=0 and AULF set

                          ; show buffer not empty.  If was output char and and turn on xmit interrupts
                          ;
0342  08AE  0001  0017    UPCnotCR    BCLR       #EMPT_W2, WB_FLG2+1(A6)  ,test and clear
0348  670A                            BOFF.S     UPCison                  ;wasn't empty before
034A  082E  0004  0015                BTST       #OUTE_W1, WB_FLG1+1(A6)  ;if output to user is disabled
0350  6602                            BON.S      UPCison                  ,then don't start xmit int

                          , interrupt will occur without sending a char
0352  6104                            BSR.S      STRTXMIT                 ,turn on interrupt

                          ; enable interrupts and exit
                          ;
0354  6000  FD44          UPCison     BRA        ENBINTS
```

```
                         ;          NOTE:    it is assumed that these routines are protected from interrupts
                         ;
                         ; STRTXMIT - start xmit interrupt process by enabling UART to interrupt
                         ;            on transmit buffer empty.
                         ; STOPXMIT - stop xmit interrupt process by disabling UART to interrupt
                         ;            on transmit buffer empty.
                         ;       Entry : D4 = unit number
                         ;
0358                     STRTXMIT
0358  7204                          MOVEQ     #XMITENB,D1              ;ENABLE XMIT INT
035A  6002                          BRA.S     SXTGETB
035C                     STOPXMIT
035C  7208                          MOVEQ     #XMITDIS,D1              ;DISABLE XMIT INT
                         ;
035E  6100  FD80         SXTGETB    BSR       GETBASE                  ;GET UART BASE ADDRESS
0362  102D  0005                    MOVE.B    CMDREGI(A5),D0           ;GET CURRENT CMD REG
0366  0200  00F3                    ANDI.B    #CLRD3D2,D0              ;CLEAR BITS D3 & D2
036A  8001                          OR.B      D1,D0                    ;DON'T CHANGE OTHER BITS
036C  1B40  0005                    MOVE.B    D0,CMDREGI(A5)           ;SAVE CHANGED CMD REG
0370  4E75                          RTS
```

```
                    ;
                    ; PutCtl - put a character in the control character buffer
                    ;       Entry : (D0) = character to put in control char buffer
                    ;                         interrupts disabled
                    ;
0372  206E  0052    PutCtl      MOVEA.L   CB_REAR(A6), A0        ;A0 = Rear pointer
0376  10C0                      MOVE.B    D0, (A0)+              ;put char in buffer and inc ptr
0378  2D48  0052                MOVE.L    A0, CB_REAR(A6)        ;put Rear pointer in save loc
037C  08AE  0001  0057          BCLR      #EMPT_CB, CB_FLAGS+1(A6) ;show not empty
0382  6004                      BRA.S     STRTXMIT               ;make sure will send character
                    ;
                    ; GetCtl - get a character from the control character buffer
                    ;       Exit  : (D3) = character to from control char buffer
                    ;                         interrupts disabled
                    ; Assumption :  The control buffer should never get full.
                    ;
0384  206E  004E    GetCtl      MOVEA.L   CB_FRONT(A6), A0       ;A0 = Front pointer
0388  1618                      MOVE.B    (A0)+, D3              ;get char from buffer and inc ptr
038A  2D48  004E                MOVE.L    A0, CB_FRONT(A6)       ;put Front pointer in save loc

038E  222E  0052                MOVE.L    CB_REAR(A6), D1        ;D1 = Rear pointer
0392  B288                      CMP.L     A0, D1                 ;Front = Rear?
0394  6204                      BHI.S     GCLexit                ;no, still more chars in buffer
                                                                 ;yes, buffer empty
0396  6100  FE0C                BSR       INITCTLB               ;init control buffer to empty

039A  4E75          GCLexit     RTS
```

```
                   ;
                   ; DCOMINT - DataCom Interrupt routine for XMIT/RCV interrupts.
                   ;
                   ; CRITICAL: if an interrupt occurs, then both the receive buffer full and the xmit
                   ;           buffer empty could be true simultaneously, so we must test both.
                   ;               However, only once thru the test then rte
                   ;           Currently the priority is reads then writes

                   ; Entry for Port 0 interrupt
                   ;
039C 48E7 FFFE     DC0INT      MOVEM.L   D0-A6,-(SP)            ;SAVE ALL REGISTERS
03A0 4DFA 068C+                LEA       Port0Data, A6          ;Address of Port 0 data
03A4 383A 0842+                MOVE.W    UnitP0, D4             ;Port 0 unit number
03A8 600C                      BRA.S     DCIcomn

                   ; Entry for Port 1 interrupt
                   ;
03AA 48E7 FFFE     DC1INT      MOVEM.L   D0-A6,-(SP)            ;SAVE ALL REGISTERS
03AE 4DFA 08DA+                LEA       Port1Data, A6          ;Address of Port 1 data
03B2 383A 0836+                MOVE.W    UnitP1, D4             ;Port 1 unit number

                   ; begin of Common port interrupt code
                   ;
03B6 6100 FD58     DCIcomn     BSR       GETBASE                ;get UART base address
03BA 1E2D 0003                 MOVE.B    STATRI(A5),D7          ;GET STATUS OF UART
                   ;
                   ; If Receive interrupt then see if should process character.
                   ;
03BE 0807 0003     DCIrcv      BTST      #S_RCVF, D7            ;TEST FOR RECEIVE BUFFER FULL
03C2 6702                      BOFF.S    DCIxmit                ;isn't, try Xmit buffer empty
03C4 6110                      BSR.S     PRcvChar               ;yes, process receive character
                   ;
                   ; Not Receive, if Transmit interrupt then see if can send character
                   ;        NOTE: THIS TESTS D7 WHICH ALLOWS US TO COME THRU HERE AFTER A READ CHECK DONE

03C6 0807 0004     DCIxmit     BTST      #S_WRTE,D7             ;XMIT BUFFER EMPTY?
03CA 6704                      BOFF.S    DCIexit                ;NO, UNKNOWN INTERRUPT - EXIT
03CC 6100 0108     DCIPX       BSR       PXMIT                  ;YES, PROCESS XMIT
03D0 4CDF 7FFF     DCIexit     MOVEM.L   (SP)+,D0-A6            ;EXIT-RESTORE REGISTERS
03D4 4E73                      RTE                              ;EXIT INTERRUPT
```

```
                        ; PRcvChar - process received character
                        ;       Entry : D7 = status register
                        ;               D4 = unit number
                        ;               A5 = UART Base address
                        ;               A6 = port's data area address
                        ;
03D6  162D  0001        PRcvChar    MOVE.B    DATAREG(A5), D3          ;GET CHAR/CLEARS INTERRUPT
                        ;;;;;;;;    BCLR      #BITD7,D3       ;don't so can send 8 bit characters

                        ; check for any errors with receive
                        ;
03DA  1007                          MOVE.B    D7, D0                   ;get status register
03DC  0200  0007                    ANDI.B    #S_ErrBits, D0           ;remove all but error bits
03E0  662E                          BNE.S     PRCerror                 ;have an error

                        ; is this a control char and (protocols enabled) and (NOT Line type)
                        ;   if yes then process com control char
                        ;
03E2  082E  0000  0012              BTST      #PROT_P2, BF_PROF(A6)    ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
03E8  6712                          BOFF.S    PRCnoctl                 ; No protocol enabled, see if can put in bu
ffer
03EA  082E  0000  0013              BTST      #Line, BF_PROF+1(A6)     ; is it a Line protocol?
03F0  660A                          BON.S     PRCnoctl                 ; Yes
03F2  0C03  0020                    CMPI.B    #' ', D3                 ; is it a control character?
03F6  6404                          BCC.S     PRCnoctl                 ; No, not in range 0 - $1F
03F8  6126                          BSR.S     PDCcontl                 ; Yes, process a possible DC control char
03FA  6722                          BEQ.S     PRCexit                  ; returns zero if processed a ctl char

                        ; check to see if input disabled or buffer full
                        ;
03FC  082E  0000  0033  PRCnoctl    BTST      #FULL_R2, RB_FLG2+1(A6)  ;is it Full?
0402  6614                          BON.S     PRClstdt                 ;Lost data error
0404  082E  0005  0031              BTST      #INPE_R1, RB_FLG1+1(A6)  ;is input disabled?
040A  660C                          BON.S     PRClstdt

                        ; put char in buffer
                        ;
040C  6000  005E                    BRA       PutChrBf

                        ; receive errors
                        ;
0410  08EE  0001  0031  PRCerror    BSET      #ERR_R1, RB_FLG1+1(A6)   ;UART error
0416  6006                          BRA.S     PRCexit
0418  08EE  0002  0033  PRClstdt    BSET      #LOST_R2, RB_FLG2+1(A6)  ;Lost data error
041E  4E75              PRCexit     RTS
```

```
                         ;
                         ; PDCcontl - check for Data Com control characters - ENQ, ETX, ACK, XON, and XOFF.
                         ;
                         ;        Entry : A6 = address of port's data area
                         ;                D3 = character
                         ;        Exit  : (NE) = char not one of the control characters
                         ;                (EQ) = char was one of the control characters
                         ;
0420  082E  0001  0013   PDCcontl     BTST      #XONXOFF, SF_PROF+1(A6)  , is it XON/XOFF protocol?
0426  662E                            BON.S     PDCLchkx                 ; yes, chk for those chars

                         ; is either ENQ/ACK or ETX/ACK   both work the same way
                         ;
0428  0C03  0006                      CMPI.B    #ACK, D3                 ,is it an ACK?
042C  671C                            BEQ.S     PDCLack                  ;yes, write is not busy now
042E  0C03  0005                      CMPI.B    #ENQ, D3                 ;is it an ENQ?
0432  6706                            BEQ.S     PDCLenq                  ;yes, see if read should go busy
0434  0C03  0003                      CMPI.B    #ETX, D3                 ,is it an ETX?
0438  6630                            BNE.S     PDCLexit                 ,no, not a control character

                         , PROCESS an ENQ or ETX - send ACK if read not busy
                         ;
043A  082E  0000  0031   PDCLenq      BTST      #BUSY_R1, RB_FLG1+1(A6)  ,is read busy?
0440  6626                            BON.S     PDCLdidit                ;yes, send ACK when clear Busy
0442  7006                            MOVEQ     #ACK, D0                 ;no, then send ACK to other side
0444  6100  FF2C         PDCLsend     BSR       PutCtl
0448  601E                            BRA.S     PDCLdidit

                         ; PROCESS an ACK and a XON - clear write busy
                         ;
044A                     PDCLxon
044A  08AE  0000  0015   PDCLack      BCLR      #BUSY_W1, WB_FLG1+1(A6)
0450  6100  FF06                      BSR       STRTXMIT                 ;start sending again
0454  6012                            BRA.S     PDCLdidit

                         ; Check for a XON or a XOFF
                         ;
0456  0C03  0011         PDCLchkx     CMPI.B    #XON, D3                 ,is it an XON?
045A  67EE                            BEQ.S     PDCLxon                  ;yes, write is not busy now
045C  0C03  0013                      CMPI.B    #XOFF, D3                ;is it an ENQ?
0460  6608                            BNE.S     PDCLexit                 ,no, not a control character

                         ; PROCESS a XOFF character - set write busy
                         ;
0462  08EE  0000  0015   PDCLxoff     BSET      #BUSY_W1, WB_FLG1+1(A6)

0468  4283               PDCLdidit    CLR.L     D3                       ;show processed
046A  4E75               PDCLexit     RTS
```

```
                   ;PutChrBf --PUT A CHARACTER INTO THE READ BUFFER AND RETURN -ADJUST COUNTERS/POINTERS AS REQ
UIRED
                   ;           COMING IN  D7- CONTAINS STATUS WORD  D3 CONTAINS CHARACTRER
                   ;                      A5 POINTS TO UART
                   ;
                   ; BUFFER HAS ENUF SPACE, JUST ADD CHARACTER
                   ;
046C 206E 0034     PutChrBf    MOVEA.L   RB_FILLP(A6), A0          , address where to put character
0470 10C3                      MOVE.B    D3, (A0)+                 ; AUTO ADJUST POINTER
0472 2D48 0034                 MOVE.L    A0, RB_FILLP(A6)          ; RESET THE FILL POINTER RB_FILLP
                   ;
                   ; Update buffer variables
                   ,
0476 226E 003C                 MOVE.L    RB_BADR(A6), A1           ;A1 = ADDRESS OF BUFFER BEGIN
047A D2EE 0040                 ADDA.W    RB_SIZE(A6), A1           ;A1 = ADDRESS OF END OF BUFFER
047E B1C9                      CMPA.L    A1, A0                    ;Is Rear pointing beyond buffer?
0480 6304                      BLS.S     PCBnowrp                  ,No, don't do wrap around
                   ;                                               ;Yes, set front = addr 1st byte of buffer
0482 2D6E 003C 0034            MOVE.L    RB_BADR(A6), RB_FILLP(A6) ;Save the new Rear pointer in rb_fillp
                   ;
0488 536E 0042     PCBnowrp    SUBQ.W    #1, RB_FREE(A6)           ;SINCE WE put in a CHAR, 1 LESS FREE SPACE

                   ; see if buffer is full  (RB_FREE is an unsigned word)
                   ;
048C 6606                      BNE.S     PCBnotfl                  ;not full, subtract sets or clears ZERO bit
048E 08EE 0000 0033            BSET      #FULL_R2, RB_FLG2+1(A6)   , BUFFER full

                   ; buffer for sure is not empty
                   ;
0494 08AE 0001 0033 PCBnotfl   BCLR      #EMPT_R2, RB_FLG2+1(A6)   ;RESET EMPTY FLAG ANYHOO

                   ; if protocols enabled and NOT Line type protocol then check buffer for hi water mark
                   ;
049A 082E 0000 0012            BTST      #PROT_P2, BF_PROF(A6)     ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
04A0 670A                      BOFF.S    PCBexit                   ; No protocol enabled, exit
04A2 082E 0008 0013            BTST      #Line, BF_PROF+1(A6)      , is it a Line protocol?
04A8 6602                      BON.S     PCBexit                   , Yes, exit
04AA 6102                      BSR.S     ChkRcvBusy                ; check for receive busy state
                   ;
04AC 4E75          PCBexit     RTS
```

```
                         ;
                         ; ChkRcvBusy - check for receive busy

                         ; is size of buffer now at or above high water mark
                         ;
04AE  322E  0040         ChkRcvBusy  MOVE.W    RB_SIZE(A6), D1          ,BUFFER SIZE (ADDRESS OF)
04B2  926E  0042                     SUB.W     RB_FREE(A6), D1          ,D1 = number of chars in buffer
04B6  B26E  004A                     CMP.W     RB_HIWA(A6), D1          , at or above hi water mark?
04BA  6402                           BCC.S     GoRcvBusy                , Yes, goto busy state
04BC  4E75               CRBTexit    RTS                                ,No, then exit
                         ;
                         ; GoRcvBusy - goto the Receive busy state
                         ; assumes interrupts are turned off
                         ,
04BE  08EE  0006  0031   GoRcvBusy   BSET      #BUSY_R1, RB_FLG1+1(A6)  ,set busy state
04C4  660E                           BON.S     GRBSexit                 ; already busy so dont send char, exit
04C6  082E  0001  0013               BTST      #XONXOFF, BF_PROF+1(A6)  ;send byte to other side saying not busy
04CC  6706                           BOFF.S    GRBSexit                 ,only if XON/XOFF protocol
04CE  7013                           MOVEQ     #XOFF, D0                ,send XOFF
04D0  6100  FEA0                     BSR       PutCtl                   ;put in control char buffer
04D4  4E75               GRBSexit    RTS                                ;if ETX/ACK or ENQ/ACK nothing else to do
```

```
                         ; PRXMIT - process transmission interrupt
                         ,           Just send the next character if possible
                         ,
                         ,           ENTRY : (A5) = UART Base address
                         ;                   (A6) = address of port's data ares
                         ;                   (D4) = unit number
                         ;                   (D7) = status byte from UART
                         ,
04D6  082E  0001  0057   PRXMIT      BTST      #EMPT_CB, CB_FLAGS+1(A6)  ;control char available?
04DC  671E                           BOFF.S    PRXgetctl                 ;yes, send it out next
                         
                         ; if NOT Busy or Buffer not empty send out next character
                         ,
04DE  082E  0000  0015               BTST      #BUSY_W1, WB_FLG1+1(A6)   ;Busy?
04E4  660C                           BON.S     PRXoff                    ;yes, turn off xmit int
04E6  082E  0001  0017               BTST      #EMPT_W2, WB_FLG2+1(A6)   ,buffer empty?
04EC  670A                           BOFF.S    PRXsend                   ;NO, send next char
                         
                         ; check for an Alternate buffer available
                         ;
04EE  6100  007C                     BSR       wChkAltBf
                         
                         ; turn off xmit ints
                         ;
04F2  6100  FE68         PRXoff      BSR       STOPXMIT
04F6  6006                           BRA.S     PRXexit
                         
                         , get next character in buffer and send out
                         ;
04F8  6110               PRXsend     BSR.S     SendNext
04FA  6002                           BRA.S     PRXexit
                         
                         ; get next control character and send it out
                         ;
04FC  6102               PRXgetctl   BSR.S     SendCtl
                         ;
04FE  4E75               PRXexit     RTS
```

```
                        1443* ;
                        1444* ; SendCtl - send next control character from control character buffer
                        1445* ;          ENTRY . (A5) = UART Base address
                        1446* ;                  (A6) = address of port's data area
                        1447* ;                  (D4) = unit number
                        1448* ;
0500  6100  FE02        1449* SendCtl     BSR       GetCtl              ,Get char and update ptrs
0504  1B43  0081        1450*             MOVE.B    D3, DATAREG(A5)     , PUSH CHARACTER OUT
0508  4E75              1451*             RTS
```

```
                          ; SendNext - Put next character in write buffer in UART transmit bufer.
                          ;           ENTRY : (A5) = UART Base address
                          ;                   (A6) = address of port's data ares
                          ;                   (D4) = unit number
                          ;
050A 206E 001C            SendNext   MOVE.L    WB_EMPTY(A6), A0       ;A0 => EMPTYING POSITION OF RD BUFFER
050E 1B58 0001                       MOVE.B    (A0)+, DATAREG(A5)     ; send out the next character
0512 2D48 001C                       MOVE.L    A0, WB_EMPTY(A6)       ;Save the new Front pointer in rb_empty

                          ; Update buffer variables
                          ;
0516 226E 0020                       MOVE.L    WB_BADR(A6), A1        ;A1 = ADDRESS OF BUFFER BEGIN
051A D2EE 0024                       ADDA.W    WB_SIZE(A6), A1        ;A1 = ADDRESS OF END OF BUFFER
051E B1C9                            CMPA.L    A1, A0                 ;Is Front pointing beyond buffer?
0520 6306                            BLS.S     SNXnowrp               ;No, don't do wrap around
                          ;                                           ;Yes, set front = addr 1st byte of buffer
0522 2D6E 0020 001C                  MOVE.L    WB_BADR(A6), WB_EMPTY(A6) ;Save the new Front pointer in wb_empty
                          ;
0528 526E 0026            SNXnowrp   ADDQ.W    #1, WB_FREE(A6)        ;SINCE WE GOT CHAR, ONE MORE FREE SPACE
052C 08AE 0000 0017                  BCLR      #FULL_W2, WB_FLG2+1(A6) ;always not full

                          ; see if buffer is empty
                          ;
0532 322E 0026                       MOVE.W    WB_FREE(A6), D1
0536 B26E 0024                       CMP.W     WB_SIZE(A6), D1        ; # OF FREE LOCATIONS - BUFFER SIZE
053A 650A                            BCS.S     SNXnotmt               ; not empty if free < size
053C 6220                            BHI.S     SNXszerr               ; size error *BUG if happens* (free > size)

053E 08EE 0001 0017                  BSET      #EMPT_W2, WB_FLG2+1(A6) ; BUFFER empty/turn off int next occurance i
n PRIMIT
0544 6126                            BSR.S     WChkAltBf              ; see if should switch to an Alternate buffe
r

                          ; if Protocols enabled and either ENQ/ACK or ETX/ACK then check
                          ;      if should send an ENQ or ETX
                          ;
0546 41EE 0012            SNXnotmt   LEA       BF_PROF(A6), A0
054A 0810 0000                       BTST      #PROT_P2, (A0)+        ;protocol enabled?
054E 671A                            BOFF.S    SNXexit                ;no, exit
0550 0810 0002                       BTST      #ENQACK, (A0)          ;ENQ/ACK protocol?
0554 6606                            BON.S     SNXcnt                 ;yes, see if should send an ENQ
0556 0810 0007                       BTST      #ETXACK, (A0)          ;ETX/ACK protocol?
055A 670E                            BOFF.S    SNXexit                ;no, exitt
055C 6030                 SNXcnt     BRA.S     CntChars               ;check if time to send ENQ or ETX

                          ; Size error - set Error flag and split
                          ;
055E 08EE 0001 0015       SNXszerr   BSET      #ERR_W1, WB_FLG1+1(A6) ;show size error
0564 08EE 0001 0017                  BSET      #EMPT_W2, WB_FLG2+1(A6) ;  mark BUFFER empty
056A 4E75                 SNXexit    RTS
```

```
                           ;
                           ; WChkAltBf - check if alternate buffer switch on write buffer
                           ;        Xmit input is automatically disabled when user calls unitstatus
                           ;        to switch buffers.
                           ;
                           ;        Entry :  A6 = address of port's data area
                           ;                 interrupts disabled
                           ;
056C  082E  0001  0017     WChkAltBf    BTST      #EMPT_W2, WB_FLG2+1(A6)  ;is buffer empty?
0572  6720                              BOFF.S    wCABexit                 ;no, can't switch
0574  45EE  0015                        LEA       WB_FLG1+1(A6), A2        ;A2 = address of write buffer flags 1
0578  0892  0002                        BCLR      #ALTBF_W1, (A2)          ;is an alternate buffer available?
057C  6716                              BOFF.S    wCABexit                 ;no, nothing to switch
                           ;
                           ; Switch buffers by making the Alternate buffer the main buffer
                           ;
057E  206E  0028                        MOVE.L    WB_ABADR(A6), A0         ;get new buffer address
0582  302E  002C                        MOVE.W    WB_ABIZE(A6), D0         ;and length
0586  6100  0464                        BSR       SetupWB                  ;switch buffer in table (doesnt use A2)
                           ;
                           ; if user is NOT controlling the Xmit input disable bit then enable
                           ;
058A  0812  0007                        BTST      #INPC_W1, (A2)           ;is user controlling input disable?
058E  6604                              BON.S     wCABexit                 ;user is controlling, exit
0590  0892  0005                        BCLR      #INPE_W1, (A2)           ;no, enable input to buffer from user
                           ;
0594  4E75                 wCABexit     RTS
```

```
                     ;
                     ; CntChars - see if sent enough characters to send out an ENQ or ETX
                     ;
                     ;       Entry :   A6 = address of port's data area
                     ;                 (A5) = UART Base address
                     ;                 (D4) = unit number
                     ;                 Protocol is either ENQ/ACK or ETX/ACK
                     ;
0596 526E 002E       CntChars   ADDQ.W   #1, WB_BENQ(A6)         ;add 1 to char count between ctl chars
059A 302E 000E                  MOVE.W   BF_BTWNEA(A6), D0       ;get max allowed between
059E B06E 002E                  CMP.W    WB_BENQ(A6), D0         ;did send max?
05A2 621A                       BHI.S    CNTexit                 ;no, then exit
                     
                     ; set max chars between last ENQ or ETX, send another and go busy until receive ACK
                     ;
05A4 7005                       MOVEQ    #ENQ, D0                ;assume send an ENQ
05A6 082E 0002 0013             BTST     #ENQACK, BF_PROF+1(A6)  ;ENQ/ACK protocol?
05AC 6602                       BNE.S    CNTenq                  ;yes
05AE 7003                       MOVEQ    #ETX, D0                ;is ETX/ACK protocol, send an ETX
05B0 6100 FDC0       CNTenq     BSR      PutCtl                  ;put char in control char buffer
05B4 426E 002E                  CLR.W    WB_BENQ(A6)             ;clear in between count
05B8 08EE 0000 0015             BSET     #BUSY_W1, WB_FLG1+1(A6) ;go write busy
05BE 4E75            CNTexit    RTS
```

```
                  ;
                  ; DCTLINT - Data Com Control interrupt service routine.
                  ;
                  ;           Makes check for both Ports.
                  ;           Calls the routine at address saved in SETVECS routine during install.
                  ;           Assumes it will clear the interrupt, toggle IOX.  The last routine in the
                  ;           chain should be the OS Level 1 routine which does turn off the interrupt.
                  ;
                  ;           Ignores the interrupt if wasn't a DataCom Control interrupt,
                  ;           therefore an Apple slot interrupt, or if NOT Line type
                  ;           handshake method.
                  ;
05C0  48E7  FFFE    DCTLINT   MOVEM.L   D0-A6,-(SP)           ;SAVE REGISTERS
05C4  4DFA  0468+             LEA       Port0Data, A6         ;do for Port 0 first
05C8  383A  091E+             MOVE.W    UnitP0, D4            ;unit number of Port 0
05CC  6114                    BSR.S     ChkLines
05CE  4DFA  06BA+             LEA       Port1Data, A6         ;do for Port 1 first
05D2  383A  0916+             MOVE.W    UnitP1, D4            ;unit number of Port 1
05D6  610A                    BSR.S     ChkLines

                  ; exit by restoring registers and then going to routine at saved address
                  ;
05D8  4CDF  7FFF              MOVEM.L   (SP)+,D0-A6           ;EXIT-RESTORE REGISTERS
05DC  2F3A  090E+             MOVE.L    SaveLvl1, -(SP)       ;take interrupt start
05E0  4E75                    RTS
```

```
                       ;
                       ; ChkLines - see if change in lines for the port$specified by A6 and D4
                       ;
                       ;       Entry : D4 = unit number for the current port
                       ;               A6 = address of data area for the current port
                       ;
05E2 4286              ChkLines    CLR.L      D6                          ;for disabling interrupts
05E4 6100 FB2A                     BSR        GETBASE                     ;setup UART base reg (A5) *2/9/83*

                       ;  See if any protocols at all and if so if any are line prots
                       ;
05E8 082E 0000 0012                BTST       #PROT_F2, BF_PROF(A6)
05EE 6730                          BOFF.S     CLNexit                     ;NO PROTOCOLS--GET OUT
                       ;
                       ; If (type of handshake () Line) then exit
                       ;
05F0 082E 0000 0013                BTST       #LINE, BF_PROF+1(A6)
05F6 6728                          BOFF.S     CLNexit                     ;NOT LINE HANDSHAKE, EXIT

                       , Determine which Line is used as Busy line Port A
                       ,
05F8 6128                          BSR.S      FINDLIN                     ;NEEDS D4 = Unit number of current port
                                                                          ;returns bit number to check in D5
                       ,
                       ;set or clear Busy depending on state of line and whether it's Busy inverted or not
                       ;
05FA 162E 0015                     MOVE.B     WB_FLG1+1(A6), D3           ;SAVE BUSY FLAG
05FE 6100 FA94                     BSR        DISINTS                     ;DISABLE INTS
0602 08EE 0000 0015                BSET       #BUSY_W1, WB_FLG1+1(A6)     ;ASSUME LINE IS BUSY = TRUE
0608 6134                          BSR.S      TSTLINE                     ;TEST LINE & INVERTED FLAG (clobbers D1 & D

2)
060A 6610                          BNE.S      CLNenbl                     ;IS BUSY
060C 08AE 0000 0015                BCLR       #BUSY_W1, WB_FLG1+1(A6)     ;not busy
                       ;
                       ; if wasn't Busy before then start up transmission process
                       ;
0612 0803 0000                     BTST       #BUSY_W1,D3                 ;TEST SAVED BUSY STATE
0616 6704                          BOFF.S     CLNenbl                     ;WASN'T BUSY
0618 6100 FD3E                     BSR        STRTXMIT                    ;START XMIT IF BUFFER NOT EMPTY
                       ;
                       ; enable interrupts
                       ;
061C 6100 FA9E         CLNenbl     BSR        ENBINTS
                       ;
0620 4E75              CLNexit     RTS
```

```
                   ;
                   ; changed function to return value in D5 *1-12-83 kb*
                   ; FINDLIN - Find which Line is used for Handshaking in Port A
                   ;            ENTRY : (D4) = unit number
                   ;                    Is a Line type protocol.
                   ;            EXIT  : (D5) = Bit # in Port A specifying line used for Busy
                   ;
0622 7A01          FINDLIN  MOVEQ    #1,D5                  ;BIT NUMBER IN PORT A CORRESPONDING TO
0624 7203                   MOVEQ    #CTSLIN,D1             ;FLAG BIT NUMBER
                   ;
                   ; Assumes that it will always find a line flag set
                   ;
0626 032E 0013     FLNLOOK  BTST     D1, BF_PROF+1(A6)      ;IS BIT SET? *1-12-83 kb*
062A 660A                   BNE.S    FLNGOT                 ;YES, D5 PORT A BIT FOR DC 0
062C 5405                   ADDQ.B   #2,D5
062E 5201                   ADDQ.B   #1,D1                  ;TRY NEXT BIT FLAG
0630 0C01 0006              CMPI.B   #DCDLIN+1, D1          ;DID LAST FLAG
0634 66F0                   BNE.S    FLNLOOK                ;NO
                   ;
                   ; if (Port 1 is unit number) then bit# := bit# + 1 - DC 1 bits in Port A are next bit up
                   ;
0636 B87A 0000+    FLNGOT   CMP.W    UnitP0, D4             ;is it Port 0?
063A 6702                   BEQ.S    FLNEXIT                ;yes, then exit
063C 5205                   ADDQ.B   #1,D5                  ;no, then  Port 1 and add 1 to bit number
063E 4E75          FLNEXIT  RTS
                   ;
                   ; changed function to receive bit number parameter in D5 *1-12-83 kb*
                   ; TSTLINE - test Port A line used for Busy and the inverted flag to show if
                   ;            Busy or NOT Busy.
                   ;            ENTRY : (D5) = bit number in Port A of Line used by Busy
                   ;            EXIT  : (NE) = Busy - D2 = $FF
                   ;                    (EQ) = NOT Busy - D2 = $00
                   ;
0640 0B39 0003 0F7F TSTLINE BTST     D5, NHIRA.L            ;Create Line Boolean
0646 56C1                   SNE      D1
0648 082E 0006 0013         BTST     #INVBUSY, BF_PROF+1(A6) ;Create Inverted Boolean
064E 56C2                   SNE      D2
0650 B302                   EOR.B    D1,D2                  ;IF RESULT IS $FF THEN BUSY
0652 4E75                   RTS
```

```
                       ; COMCLR - UNITCLEAR
                       ,             Initialize Buffers to empty.  Initialize Communications control
                       ;             variables.  Initialize UART from Printer Control Table.
                       ;
0654  6100  FA40         COMCLR      BSR        DISINTS                   ,DISABLE INTERRUPTS
0658  6100  FA8A                     BSR        SETUART                   ,INIT UART FROM CONSTANTS & TABLE
065C  6108                           BSR.S      InitBufs                  ;initialize read and write buffers to empty
065E  6158                           BSR.S      ClrBusy                   ;clear busy flags
0660  6100  FA5A                     BSR        ENBINTS                   ;ENABLE INTERRUPTS
0664  4E75                           RTS
                       ;
                       ; InitBufs - initialize read, write and control buffers to empty
                       ;              enable out and in bound on both buffers, remove alt buffers
                       ;       Exit : (D0) = old busy flag for read buffer
                       ;                (D1) = old busy flag for write buffer
                       ;
0666  2D6E  003C  0034   InitBufs    MOVE.L     RB_BADR(A6), RB_FILLP(A6) ;initialize front and
066C  2D6E  003C  0038               MOVE.L     RB_BADR(A6), RB_EMPTY(A6) ; rear pointers
0672  3D6E  0040  0042               MOVE.W     RB_SIZE(A6), RB_FREE(A6)  ,show count as all free
0678  426E  0032                     CLR.W      RB_FLG2(A6)               ; reset AutoLf, send LF, Full and Lost
067C  08EE  0001  0033               BSET       #EMPT_R2, RB_FLG2+1(A6)   ; BUFFER IS EMPTY
0682  302E  0030                     MOVE.W     RB_FLG1(A6), D0           ;GET old busy flag
0686  426E  0030                     CLR.W      RB_FLG1(A6)               ;reset all flags

068A  2D6E  0020  0018               MOVE.L     WB_BADR(A6), WB_FILLP(A6) ;initialize front and
0690  2D6E  0020  001C               MOVE.L     WB_BADR(A6), WB_EMPTY(A6) ; rear pointers
0696  3D6E  0024  0026               MOVE.W     WB_SIZE(A6), WB_FREE(A6)  ;show count as all free
069C  426E  0016                     CLR.W      WB_FLG2(A6)               ; reset send LF, Full and Lost
06A0  08EE  0004  0017               BSET       #AULF_W2, WB_FLG2+1(A6)   ; DO AUTO LF and *kb 1/5/83*
06A6  08EE  0001  0017               BSET       #EMPT_W2, WB_FLG2+1(A6)   ; BUFFER IS EMPTY
06AC  322E  0014                     MOVE.W     WB_FLG1(A6), D1           ;GET old busy flag
06B0  426E  0014                     CLR.W      WB_FLG1(A6)               ;reset all flags

06B4  6000  FAEE                     BRA        INITCTLB                  ,init ctl char buffer
                       ,
                       ; ClrBusy - if Read buffer was busy then send out NOT busy state
                       ;             ignore write busy for now
                       ;
                       ;       Entry : (D0) = old busy flag for read buffer
                       ;                 (D1) = old busy flag for write buffer
                       ;
06B8  0800  0000         ClrBusy     BTST       #BUSY_R1, D0
06BC  6704                           BOFF.S     CBSexit                   ;may have to check if have
06BE  6100  FBB2                     BSR        GoUnBusy                  ;protocols and not line type
06C2  4E75               CBSexit     RTS
```

```
                                  ;
                                  ; COMBSY - UNITBUSY
                                  ;             PASCAL BOOLEAN TRUE RETURNED IN D0 IF THERE ARE ANY CHARACTERS IN READ BUFFER
                                  ;
        06C4                      COMBSY
        06C4  082E  0001  0033               BTST      #EMPT_R2, RB_FLG2+1(A4)
        06CA  57C0                           SEQ       D0                       ,IF BIT NOT SET THEN = 0; CHARACTRERS EXIST
D0 =111111          .
        06CC  0200  0001                     ANDI.B    #TRUE,D0                 ;CONVERT FROM BOOLEAN TO PASCAL BOOLEAN-
        06D0  4E75                           RTS
                                  ,
                                  ; COMUNMT - UNITUNMOUNT
                                  ;             Turnoff interrupt capabilities of COMM  driver.
                                  ;             Restore vectors.
                                  ;
        06D2  6100  F9C2          COMUNMT    BSR       DISINTS                  ;DISABLE INTERRUPTS
                                  ;
        06D6  6100  FA38                     BSR       GETBASE                  ;GET UART BASE
        06DA  187C  0002  0005               MOVE.B    #TURNOFF,CMDREG1(A5)     ;TURNOFF UART
                                  ;
                                  ; have vectors point to a RTE instruction
                                  ;
        06E0  41F8  0070                     LEA       VEC4.W, A0               ;assume it is Port 0
        06E4  43F8  0068                     LEA       VEC2.W, A1
        06E8  45FA  001A+                    LEA       THERTE, A2               ;address of the RTE instruction
        06EC  B87A  07FA+                    CMP.W     UnitP0, D4               ;is it Port 0?
        06F0  6702                           BEQ.S     CUMisP0                  ;yes, change level 4
        06F2  C348                           EXG       A0, A1                   ;no, change level 2
        06F4  208A                CUMisP0    MOVE.L    A2, (A0)                 ;set vector to point at RTE
                                  ;
                                  ; if both vectors point at RTE then set level 1 to saved address
                                  ;
        06F6  B388                           CMPM.L    (A0)+, (A1)+             ;must do post inc.
        06F8  6606                           BNE.S     CUMdiff                  ;different so not both RTE
        06FA  21FA  07F0+ 0064               MOVE.L    SaveLvl1, VEC1.W         ;restore from saved area
                                  ;
                                  ; Restore Interrupts
                                  ;
        0700  6000  F9BA          CUMdiff    BRA       ENBINTS

                                  ; THE RTE INSTRUCTION
                                  ;
        0704  4E73                THERTE     RTE
```

```
                 ;
                 ; CONST - UNITSTATUS
                 ; call the Table change or buffer free Functions
                 ;
0706 0C42 0014   CONST      CMPI.W    #TBLSTATE,D2           ;VALID FUNCTION CODE
070A 6210                   BHI.S     CSTERR                 ;NO
070C 3013                   MOVE.W    (A3),D0                ;GET PARAMETER
070E 43FA 0010+             LEA       CSTTBL,A1              ;TURN THE FUNCTION CODE INTO
0712 E34A                   LSL.W     #1,D2                  ;AN INDEX TO THE FUNCTION
0714 3431 2000              MOVE.W    0(A1,D2.W),D2
0718 4EF1 2000              JMP       0(A1,D2.W)             ;DO FUNCTION
                 ;
                 ; Invalid Function Code Error
                 ;
071C 7E38        CSTERR     MOVEQ     #INVFNC,D7
071E 4E75                   RTS
                 ;
                 ; THE COM DRIVER STATUS JUMP TABLE
                 ;
                 ; functions compatible with old printer driver

0720 002E        CSTTBL     DATA.W    STWBUF-CSTTBL          ;WRITE BUFFER FREE SPACE
0722 0040                   DATA.W    STBAUD-CSTTBL          ;SET BAUD RATE
0724 0050                   DATA.W    STPRITY-CSTTBL         ;SET PARITY
0726 0038                   DATA.W    STRBUF-CSTTBL          ;READ BUFFER FREE SPACE
0728 0060                   DATA.W    STWRDSZ-CSTTBL         ;SET WORD SIZE
072A 007E                   DATA.W    STHNDSK-CSTTBL         ;SET HANDSHAKE METHOD
072C 0164                   DATA.W    STBFSTS-CSTTBL         ;TELL BUFFER CONTROL STATUS

                 ; new functions

072E 00D2                   DATA.W    STRDSTS-CSTTBL         ;TELL READ STATUS
0730 0122                   DATA.W    STWTSTS-CSTTBL         ;TELL WRITE STATUS
0732 0088                   DATA.W    STRDHI-CSTTBL          ;SET READ BUFFER HI WATER MARK
0734 00BE                   DATA.W    STRDLO-CSTTBL          ;SET READ BUFFER LOW WATER MARK
0736 0180                   DATA.W    STOUTRD-CSTTBL         ;TOGGLE OUTBOUND READ
0738 01C0                   DATA.W    STINRD-CSTTBL          ;TOGGLE INBOUND READ
073A 01D6                   DATA.W    STOUTWT-CSTTBL         ;TOGGLE OUTBOUND WRITE
073C 01F2                   DATA.W    STINWT-CSTTBL          ;TOGGLE INBOUND WRITE
073E 0232                   DATA.W    BWBCHR-CSTTBL          ;TELL #CHARS IN WRITE BUFFER
0740 023E                   DATA.W    BRBCHR-CSTTBL          ;TELL #CHARS IN READ BUFFER
0742 00C4                   DATA.W    STATOLF-CSTTBL         ;TOGGLE auto LineFeed flag
0744 00CC                   DATA.W    STBENQ-CSTTBL          ;SET number of chars between ENQ's
0746 024A                   DATA.W    STRDALTBF-CSTTBL       ;SET Read Alternate buffer
0748 0292                   DATA.W    STWTALTBF-CSTTBL       ;SET Write Alternate bufferen ENQ's
```

```
074A 6000 F94A    STCallDis   BRA       DISINTS                ;DISABLE INTERRUPTS
                  ;
                  ; STWBUF - Return to the user the Free space in the write buffer
                  ;
074E 61FA         STWBUF      BSR.S     STCallDis              ;DISABLE INTERRUPTS
0750 36AE 0026                MOVE.W    WB_FREE(A6), (A3)      ;WRITE BUFFER FREE SPACE
0754 6000 F966    STCallEnb   BRA       ENBINTS                ;ENABLE INTERRUPTS
                  ;
                  ; STRBUF - Return to the user the Free space in the READ buffer
                  ;
0758 61F0         STRBUF      BSR.S     STCallDis              ;DISABLE INTERRUPTS
075A 36AE 0042                MOVE.W    RB_FREE(A6), (A3)      ;WRITE BUFFER FREE SPACE
075E 60F4                     BRA.S     STCallEnb              ;ENABLE INTERRUPTS
                  ;
                  ; STBAUD - Set the Baud Rate
                  ;
0760 0C40 0006    STBAUD      CMPI.W    #MAXBAUD,D0            ;IS IT A VALID PARAMETER
0764 6234                     BHI.S     SETERR                 ;NO
                  ;
0766 41EE 000B                LEA       BF_RDBD(A6), A0        ;WHERE TO PUT VALUE
076A 43FA 02AC+               LEA       BAUDCNV,A1             ;CONVERSION ARRAY
076E 601E                     BRA.S     SAVPARM                ;SAVE CONVERTED PARAMETER
                  ;
                  ; STPRITY - Set the Parity
                  ;
0770 0C40 0004    STPRITY     CMPI.W    #MAXPRTY,D0            ;IS IT A VALID PARAMETER
0774 6224                     BHI.S     SETERR                 ;NO
                  ;
0776 41EE 000C                LEA       BF_PART(A6), A0        ;WHERE TO PUT VALUE
077A 43FA 02A3+               LEA       PRTYCNV,A1             ;CONVERSION ARRAY
077E 600E                     BRA.S     SAVPARM                ;SAVE CONVERTED PARAMETER
                  ;
                  ; STWRDSZ - Set the word size to transmit (7 or 8)
                  ;
0780 0C40 0001    STWRDSZ     CMPI.W    #MAXWRDS,D0            ;IS IT A VALID PARAMETER
0784 6214                     BHI.S     SETERR                 ;NO
                  ;
0786 41EE 000D                LEA       BF_WRDS(A6), A0        ;WHERE TO PUT VALUE
078A 1080                     MOVE.B    D0,(A0)                ;PUT IN WORD SIZE VALUE
078C 6004                     BRA.S     RSTUART                ;RESET UART FROM TABLE
                  ;
                  ; common code to STBAUDR, STPRITY, STWRDSZ, STDTACOM, & STHNDSK
                  ;
078E 10B1 0000    SAVPARM     MOVE.B    0(A1,D0.W),(A0)        ;SAVE CONVERTED PARAMETER
                  ;
0792 6186         RSTUART     BSR.S     STCallDis              ;DISABLE INTERRUPTS
0794 6100 F94E    RSTUART1    BSR       SETUART                ;SETUP UART FROM TABLE
0798 608A                     BRA.S     STCallEnb              ;ENABLE INTERRUPTS
                  ;
                  ; Invalid Parameter error
                  ;
079A 7E36         SETERR      MOVEQ     #INVPRM,D7
079C 4E75                     RTS
```

```
                          ;
                          ; STHNDSK - Set Handshake type.  Convert parameter into the flags and put these
                          ;           flag values into the Printer Control Table.  Don't need to reset
                          ;           UART.
                          ;
079E                      STHNDSK
079E  0C40  0009                        CMPI.W    #MAXHNDS,D0            ;IS IT A VALID PARAMETER
07A2  62F6                              BHI.S     SETERR                 ;NO
                          ;
07A4  43FA  027E+                       LEA       HNDSCNV,A1             ;CONVERSION ARRAY
07A8  1D71  0000  0013                  MOVE.B    0(A1,D0.W), BF_PROF+1(A6) ;move new flags into flag byte

                          ; see if user disabled all protocols
                          ;
07AE  08EE  0000  0012                  BSET      #PROT_P2, BF_PROF(A6)   ;assume have a protocol
07B4  4A2E  0013                        TST.B     BF_PROF+1(A6)          ;IF zero then no protocols
07B8  6608                              BNE.S     SHDchkEA               ;see if ETX/ACK or ENQ/ACK
07BA  08AE  0000  0012                  BCLR      #PROT_P2, BF_PROF(A6)   ;show no protocol
07C0  6014                              BRA.S     SHDexit


                          ;
                          ;
07C2  082E  0007  0013    SHDchkEA      BTST      #ETXACK, BF_PROF+1(A6)   ;is it ETX/ACK?
07C8  6608                              BON.S     SHDzero                 ;yes, zero char count
07CA  082E  0002  0013                  BTST      #ENQACK, BF_PROF+1(A6)   ;is it ENQ/ACK?
07D0  6704                              BOFF.S    SHDexit                 ;no, exit
07D2  426E  002E          SHDzero       CLR.W     WB_BENQ(A6)             ;clr cnt of chars between ENQ's or ETX's
                          ;
07D6  4E75                SHDexit       RTS
```

```
                          ;
                          ;STRDHI  -SET THE READ BUFFER HIGH WATER MARK
                          ;
07D8  3D53  004A          STRDHI      MOVE.W     (A3), RB_HIWA(A6)
07DC  4E75                            RTS
                          ;
                          ;STRDLO  -SET THE READ BUFFER LOW WATER MARK
                          ;
07DE  3D53  004C          STRDLO      MOVE.W     (A3), RB_LOWA(A6)
07E2  4E75                            RTS
                          ;
                          ; STATLF - toggle the Auto LineFeed flag
                          ;
07E4  086E  0004  0017    STATOLF     BCHG       #AULF_V2, WB_FLG2+1(A6)  ;flip the bit
07EA  4E75                            RTS
                          ;
                          ; STBENQ - set the number of chars between ENQ's or ETX's
                          ;
07EC  3D53  000E          STBENQ      MOVE.W     (A3), SF_BTWNEA(A6)
07F0  4E75                            RTS
```

```
                     ; STRDSTS - GET THE READ BUFFER STATUS
                     ;         ParameterBlock = record
                     ;                           BufferSize     : integer;
                     ;                           FreeSpace      : integer;
                     ;                           HiWater        : integer;
                     ;                           LowWater       : integer;
                     ;                           InputDisabled  : byte;    (true = 1, false = 0)
                     ;                           OutputDisabled : byte;    (true = 1, false = 0)
                     ;                           LostData       : byte;    (true = 1, false = 0)
                     ;                           AltBufferAvail : byte;    (true = 1, false = 0)
                     ;                           AltBufferAddr  : pointer; (0 if AltBufferAvail false)
                     ;                           AltBufferSize  : integer; (0 if AltBufferAvail false)
                     ;                         end;
                     ;
07F2 36EE 0040       STRDSTS     MOVE.W     RB_SIZE(A6), (A3)+       ;get buffer size
07F6 36EE 0042                   MOVE.W     RB_FREE(A6), (A3)+       ;get free space byte count
07FA 36EE 004A                   MOVE.W     RB_HIWA(A6), (A3)+       ;get hi water byte count
07FE 36EE 004C                   MOVE.W     RB_LOWA(A6), (A3)+       ;get low water byte count

                     ; get the flags and make byte Pascal booleans
                     ;
0802 082E 0005 0031              BTST       #INPE_R1, RB_FLG1+1(A6)  ;is PORT to BUFFER disabled?
0808 612E                        BSR.S      MAKEBOOL
080A 082E 0004 0031              BTST       #OUTE_R1, RB_FLG1+1(A6)  ;is BUFFER to USER disabled?
0810 6126                        BSR.S      MAKEBOOL
0812 08AE 0002 0033              BCLR       #LOST_R2, RB_FLG2+1(A6)  ;has any data been lost?
0818 611E                        BSR.S      MAKEBOOL

                     ; If have an Alt buffer then return it's ADDRESS AND SIZE
                     ;
081A 082E 0002 0031              BTST       #ALTBF_R1, RB_FLG1+1(A6)
0820 670E                        BOFF.S     RDSTnone
0822 16FC 0001                   MOVE.B     #1, (A3)+                 ;set Alt buffer boolean
0826 26EE 0044                   MOVE.L     RB_ABADR(A6), (A3)+       ;get Alternate buffer Address
082A 36AE 0048                   MOVE.W     RB_ASIZE(A6), (A3)        ;get Alternate buffer size
082E 6006                        BRA.S      RDSTexit

0830 421B            RDSTnone    CLR.B      (A3)+                     ;no Alternate buffer available
0832 429B                        CLR.L      (A3)+                     ;so NIL pointer for address
0834 4253                        CLR.W      (A3)                      ;and zero bytes size

0836 4E75            RDSTexit    RTS
                     ;
                     ; MAKEBOOL - make Pascal boolean from zero flag
                     ;
0838 56C0            MAKEBOOL    SNE        D0              ;D0.B = $FF if zero flag clear
083A 0200 0001                   ANDI.B     #TRUE, D0       ;turn to Pascal boolean (1 = true)
083E 16C0                        MOVE.B     D0, (A3)+       ;save in parameter block
0840 4E75                        RTS
```

```
                       ; STWTSTS - GET THE WRITE BUFFER STATUS
                       ;       ParameterBlock = record
                       ;                               BufferSize     : integer;
                       ;                               FreeSpace      : integer;
                       ; FROM BUFFER CNTRL TBL -> CharsBtwnENQs  : integer;
                       ;                               InputDisabled  . byte;    (true = 1, false = 0)
                       ;                               OutputDisabled : byte;    (true = 1, false = 0)
                       ;                               AutoLineFeed   : byte;    (true = 1, false = 0)
                       ;                               AltBufferAvail : byte;    (true = 1, false = 0)
                       ;                               AltBufferAddr  : pointer, (0 if AltBufferAvail false)
                       ;                               AltBufferSize  : integer; (0 if AltBufferAvail false)
                       ;                             end,
                       ;
0842  36EE  0024       STWTSTS    MOVE.W    WB_SIZE(A6), (A3)+       ;get buffer size
0846  36EE  0026                  MOVE.W    WB_FREE(A6), (A3)+       ;get free space byte count
084A  36EE  000E                  MOVE.W    BF_BTWNEA(A6), (A3)+     ,get maximum number of chars between ENQ's o
r ETX's

                       ; get the flags and make byte Pascal booleans
                       ;
084E  082E  0005  0015            BTST      #INPE_W1, WB_FLG1+1(A6)  ;is USER to BUFFER disabled?
0854  61E2                        BSR.S     MAKEBOOL
0856  082E  0004  0015            BTST      #OUTE_W1, WB_FLG1+1(A6)  ;is BUFFER to PORT disabled?
085C  61DA                        BSR.S     MAKEBOOL
085E  082E  0004  0017            BTST      #AULF_W2, WB_FLG2+1(A6)  ;is Auto LineFeed mode on?
0864  61D2                        BSR.S     MAKEBOOL

                       ; If have an Alt buffer then return it's ADDRESS AND SIZE
                       ;
0866  082E  0002  0015            BTST      #ALTBF_W1, WB_FLG1+1(A6)
086C  670E                        BOFF.S    WTSTnone
086E  16FC  0001                  MOVE.B    #1, (A3)+                ;set Alt buffer boolean
0872  26EE  0028                  MOVE.L    WB_ABADR(A6), (A3)+      ;get Alternate buffer Address
0876  36AE  002C                  MOVE.W    WB_ASIZE(A6), (A3)       ;get Alternate buffer size
087A  6006                        BRA.S     WTSTexit

087C  421B             WTSTnone   CLR.B     (A3)+                    ;no Alternate buffer available
087E  429B                        CLR.L     (A3)+                    ;so NIL pointer for address
0880  4253                        CLR.W     (A3)                     ;and zero bytes size

0882  4E75             WTSTexit   RTS
```

```
                 ; STBFSTS - Return to the user in the parameter block the state of the Buffer Control Table.
                 ;       ParameterBlock = record
                 ;                           BaudRate  : integer; ;(range = 0..6)
                 ;                           Parity    : integer; ;(range = 0..4)
                 ;                           DataCom   : integer; ;(range = 0..1)
                 ;                           WordSize  : integer; ;(range = 0..1)
                 ;                           HandShake : integer; ;(range = 0..9)
                 ;                         end;
                 ;
0884 4281         STBFSTS    CLR.L     D1                      ;MAKE SURE NO GARBAGE IN REGISTER
                 ;
                 ; GET BAUD RATE
                 ;
0886 303C 0006               MOVE.W    #MAXBAUD,D0             ;MAX BAUD RATE PARAMETER VALUE
088A 122E 000B               MOVE.B    BF_RDBD(A6), D1         ;CURRENT TABLE VALUE
088E 41FA 0188+              LEA       BAUDCNV,A0              ;CONVERT TO INTEGER RANGE
0892 612C                    BSR.S     GETVAL
                 ;
                 ; GET PARITY
                 ;
0894 303C 0004               MOVE.W    #MAXPRTY,D0             ;MAX PARITY PARAMETER VALUE
0898 122E 000C               MOVE.B    BF_PART(A6), D1         ;CURRENT TABLE VALUE
089C 41FA 0181+              LEA       PRTYCNV,A0              ;CONVERT TO INTEGER RANGE
08A0 611E                    BSR.S     GETVAL
                 ;
                 ; GET DATACOM - BASED ON D4 and the SAVED UNIT NUMBER
                 ;
08A2 4281                    CLR.L     D1                      ;assume is Port 0
08A4 B87A 0642+              CMP.W     UnitP0, D4              ;is Port 0?
08A8 6702                    BEQ.S     SBFSisP0                ;yes
08AA 7201                    MOVEQ     #1, D1                  ;no, show as Port 1
08AC 36C1         SBFSisP0   MOVE.W    D1, (A3)+               ;save parameter
                 ;
                 ; GET WORD SIZE
                 ;
08AE 122E 000D               MOVE.B    BF_WRDS(A6), D1
08B2 36C1                    MOVE.W    D1, (A3)+
                 ;
                 ; GET HANDSHAKE
                 ;
08B4 303C 0009               MOVE.W    #MAXHNDS,D0             ;MAX HANDSHAKE PARAMETER VALUE
08B8 122E 0013               MOVE.B    BF_PROF+1(A6),D1        ;CURRENT TABLE VALUE
08BC 41FA 0166+              LEA       HNDSCNV,A0              ;CONVERT TO INTEGER RANGE
                 ;
                 ; GET PARAMETER VALUE AN PUT IN PARAMETER BLOCK
                 ;
08C0 B230 0000    GETVAL     CMP.B     0(A0,D0.W), D1          ;SEE WHICH CONVERSION VALUE = CURRENT VALUE
08C4 57C8 FFFA               DBEQ      D0, GETVAL              ;THE INDEX OF ONE = IS THE PARAMETER VALUE
TO
08C8 36C0                    MOVE.W    D0, (A3)+               ;RETURN TO USER IN PARAMETER BLOCK
08CA 4E75                    RTS
```

```
08CC 6000 F7C8          DoDisInt    BRA       DISINTS                 ;DISABLE INTERRUPTS
                        ;
                        ; STOUTRD -- TOGGLE OUTBOUND RECEIVE DISABLE   (BUFFER TO USER)
                        ;
08D0 61FA               STOUTRD     BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08D2 086E 0006 0031                 BCHG      #OUTC_R1, RB_FLG1+1(A6)
08D8 086E 0004 0031                 BCHG      #OUTE_R1, RB_FLG1+1(A6)
08DE 6012                           BRA.S     DoEnbInt                ;enable interrupts
                        ;
                        ; STINRD -- TOGGLE INBOUND RECEIVE DISABLE  (PORT TO BUFFER)
                        ;
08E0 61EA               STINRD      BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08E2 41EE 0031                      LEA       RB_FLG1+1(A6), A0       ;address of flags
08E6 0850 0007                      BCHG      #INPC_R1, (A0)          ;user currently controlling?
08EA 6604                           BNE.S     INRDenb                 ;yes, then enable
08EC 6148                           BSR.S     DisRcvIn                ;no, disable input
08EE 6002                           BRA.S     DoEnbInt
08F0 6136               INRDenb     BSR.S     EnbRcvIn

08F2 6000 F7C8          DoEnbInt    BRA       ENBINTS                 ;ENABLE INTERRUPTS
                        ;
                        ; STOUTWT -- TOGGLE OUTBOUND TRANSMIT DISABLE  (BUFFER TO PORT)
                        ;
08F4 61D4               STOUTWT     BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08F8 086E 0006 0015                 BCHG      #OUTC_W1, WB_FLG1+1(A6) ;toggle user controlling
08FE 086E 0004 0015                 BCHG      #OUTE_W1, WB_FLG1+1(A6) ;and enable/disable flag
0904 6606                           BNE.S     OTWToff                 ;now disabled, turn off xmit int
0906 6100 FA50                      BSR       STRTXMIT                ;enable xmit int
090A 60E6                           BRA.S     DoEnbInt
090C 6100 FA4E          OTWToff     BSR       STOPXMIT                ;disable xmit ints
0910 60E0                           BRA.S     DoEnbInt                ;enable interrupts
                        ;
                        ; STINWT -- TOGGLE INBOUND TRANSMIT DISABLE  (USER TO BUFFER)
                        ;
0912 61B8               STINWT      BSR.S     DoDisInt                ;DISABLE INTERRUPTS
0914 41EE 0015                      LEA       WB_FLG1+1(A6), A0       ;address of flags
0918 0850 0007                      BCHG      #INPC_W1, (A0)          ;toggle user controlling
091C 0810 0002                      BTST      #ALTBF_W1, (A0)         ;if got an alt buffer
0920 66D0                           BNE.S     DoEnbInt                ;then already set, let it enable
0922 0850 0005                      BCHG      #INPE_W1, (A0)          ;else toggle it
0926 60CA                           BRA.S     DoEnbInt                ;enable interrupts
```

```
                         ;
                         ; EnbRcvIn - Enable input receive
                         ;       Entry : A6 = address of ports data area
                         ;               A0 = address of read buffer flag 1 low byte
                         ;               interrupts disabled
                         ;
0928  0810  0002         EnbRcvIn    BTST      #ALTBF_R1, (A0)          ;alternate buffer available?
092C  6604                           BON.S     ERIchkprt                ;yes then let switch enable RCV
092E  0890  0005                     BCLR      #INPE_R1, (A0)           ;NO, enable input

                         ; see if should tell other side not BUSY
                         ;
0932  6000  F910         ERIchkprt   BRA       ChkProto
                         ;
                         ; DisRcvIn
                         ;       Entry : A6 = address of ports data area
                         ;               A0 = address of read buffer flag 1 low byte
                         ;               interrupts disabled
                         ;
0936  08D0  0005         DisRcvIn    BSET      #INPE_R1, (A0)           ,disable input
093A  6614                           BON.S     DRIexit                  ;if was off then don't go busy again

                         ; if protocols enabled and NOT Line type protocol then go busy
                         ;
093C  082E  0000  0012               BTST      #PROT_P2, BF_PROF(A6)    ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
0942  670C                           BOFF.S    DRIexit                  , No protocol enabled, exit
0944  082E  0000  0013               BTST      #Line, BF_PROF+1(A6)     ; is it a Line protocol?
094A  6604                           BON.S     DRIexit                  ; Yes, exit
094C  6100  FB70                     BSR       GoRcvBusy                , go busy

0950  4E75               DRIexit     RTS
```

```
                    ;
                    ; BWBCHR -  GET Number of characters  IN the WRITE BUFFER
                    ;
0952  302E  0024    BWBCHR      MOVE.W    WB_SIZE(A6), D0         ;SIZE IN D1
0956  906E  0026                SUB.W     WB_FREE(A6), D0         ; SIZE - FREE = Number of CHARS
095A  3680                      MOVE.W    D0, (A3)                ; return to user amount
095C  4E75                      RTS
                    ;
                    ; BRBCHR -  GET Number of characters  IN the READ BUFFER
                    ;
095E  302E  0040    BRBCHR      MOVE.W    RB_SIZE(A6), D0         ;SIZE IN D1
0962  906E  0042                SUB.W     RB_FREE(A6), D0         ; SIZE - FREE = Number of CHARS
0966  3680                      MOVE.W    D0, (A3)                ; return to user amount
0968  4E75                      RTS
```

```
                        ;
                        ; STRDALTBF - set Alternate Buffer for Read
                        ;
096A 6100 F72A          STRDALTBF   BSR       DISINTS                 ; disable interrupts
096E 6138                           BSR.S     GetAltBuf               ; get address and size user passed
0970 660A                           BNE.S     RDABsz                  ; addr ok, check size
0972 41EE 005C                      LEA       RDBUF(A6), A0           ; they were zero so use default
0976 303C 0100                      MOVE.W    #RBFLEN, D0             ; read buffer
097A 6008                           BRA.S     RDABok

097C 4A40               RDABsz      TST.W     D0                      , check size, is it negative?
097E 6A04                           BPL.S     RDABok                  ; no
0980 7E36               STBFerr     MOVEQ     #INVPRM, D7             ; yes, invalid parameter
0982 6020                           BRA.S     RDABexit

                        ; got buffer address and length
                        ;
0984 082E 0001 0033     RDABok      BTST      #EMPT_R2, RB_FLG2+1(A6)  ; current buffer empty?
098A 6616                           BON.S     RDABswtch               ; yes, then use user's buffer

                        ; buffer isn't empty so wait till empty to switch
                        ;
098C 2D48 0044                      MOVE.L    A0, RB_ABADR(A6)        ;save address in alt buffer adr
0990 3D40 0048                      MOVE.W    D0, RB_ASIZE(A6)        ;and length in alt buffer size
0994 41EE 0031                      LEA       RB_FLG1+1(A6), A0       ;DisRcvIn needs A0 -> flag byte
0998 08D0 0002                      BSET      #ALTBF_R1, (A0)         ;alt buffer available true
099C 6100 FF98                      BSR       DisRcvIn                ;disable input and see if should go busy
09A0 6002                           BRA.S     RDABexit                ; exit

                        ; EMPTY SO MAKE NEW the current buffer
                        ;
09A2 6150               RDABswtch   BSR.S     SetupRB

09A4 6000 F716          RDABexit    BRA       ENBINTS
                        ;
                        ; GetNewBuf - from user's parameter block get the Alt buffer
                        ;                  address and size.
                        ;       EXIT   . D0 = alt buffer length
                        ;                  D1 = 0
                        ;                  A0 = alt buffer address
                        ;                  (EQ) = use default
                        ;                  (NE) = use A0 and D0
                        ;
09A8 205B               GetAltBuf   MOVEA.L   (A3)+, A0
09AA 301B                           MOVE.W    (A3)+, D0
09AC 4281                           CLR.L     D1
09AE B288                           CMP.L     A0, D1
09B0 4E75               GABFexit    RTS
```

```
                        ; STWTALTBF - set Alternate Buffer for Write
                        ;
09B2  6100  F6E2        STWTALTBF   BSR       DISINTS                  ; disable interrupts
09B6  61F0                          BSR.S     GetAltBuf                ;get user's buffer address and size
09B8  660A                          BNE.S     WTABsz                   ; addr good chk size
09BA  41EE  015C                    LEA       WRTBUF(A6), A0           ; they were zero so use default
09BE  303C  0100                    MOVE.W    #WBFLEN, D0              ; write buffer
09C2  6004                          BRA.S     WTABok
                        
09C4  4A40              WTABsz      TST.W     D0                       ; check size, is it negative?
09C6  6BB8                          BMI.S     STBFerr                  ; no
                        
                        ; got buffer address and length
                        ;
09C8  082E  0001  0017  WTABok      BTST      #EMPT_W2, WB_FLG2+1(A6)  ; current buffer empty?
09CE  6616                          BNE.S     WTABswtch                ; yes, then use user's buffer
                        
                        ; buffer isn't empty so wait till empty to switch
                        ;
09D0  2D48  0028                    MOVE.L    A0, WB_ABADR(A6)         ;save address in alt buffer adr
09D4  3D40  002C                    MOVE.W    D0, WB_ASIZE(A6)         ;and length in alt buffer size
09D8  08EE  0005  0015              BSET      #INPE_W1, WB_FLG1+1(A6)  ;disable input
09DE  08EE  0002  0015              BSET      #ALTBF_W1, WB_FLG1+1(A6) ;alt buffer available true
09E4  6002                          BRA.S     WTABexit                 ; exit
                        
                        ; EMPTY SO MAKE NEW the current buffer
                        ;
09E6  6104              WTABswtch   BSR.S     SetupWB
                        
09E8  6000  F6D2        WTABexit    BRA       ENBINTS
                        ;
                        ; SetupRB - put the alternate buffer info in the Read Buffer Control Table
                        ; SetupWB - put the alternate buffer info in the Write Buffer Control Table
                        ;       Entry : D0 = alternate buffer size
                        ;               A0 = alternate buffer address
                        ;
09EC  43EE  0018        SetupWB     LEA       WB_FILLP(A6), A1
09F0  99CC                          SUBA.L    A4,A4
09F2  6008                          BRA.S     STUPgo
                        
09F4  43EE  0034        SetupRB     LEA       RB_FILLP(A6), A1
09F8  49EE  005C                    LEA       RDBUF(A6), A4
                        
                        ; move the buffer address into the Front, Rear, and buffer pointers
                        ;
09FC  22C8              STUPgo      MOVE.L    A0, (A1)+                ;set the fill (Front) pointer
09FE  22C8                          MOVE.L    A0, (A1)+                ;set the empty (Rear) pointer
0A00  22C8                          MOVE.L    A0, (A1)+                ;set the buffer pointer
                        
                        ; move the size into the buffer size and free space counter
                        ;
0A02  32C0                          MOVE.W    D0, (A1)+                ;set the size
0A04  32C0                          MOVE.W    D0, (A1)+                ;set the free space available
                        
```

```
                       2239* ; see if should set water marks for read buffer
                       2240* ;
0A04  B9C8             2241*              CMPA.L     A0, A4
0A08  660C             2242*              BNE.S      STUPexit
0A0A  3D7C  0085  004A 2243*              MOVE.W     #MAXRHI, RB_HIWA(A6)
0A10  3D7C  0050  004C 2244*              MOVE.W     #MAXRLO, RB_LOWA(A6)
0A16  4E75             2245* STUPexit     RTS
```

```
                         ;
                         ; constant data area
                         ;
                         ; Conversion arrays for Set functions of Unitstatus
                         ;
0A18  06 07 08 0A 0C 0E  BAUDCNV     DATA.B     6,7,8,$A,$C,$E,$F         ;BAUD RATE
0A1E  0F
                         ; 6=300,7=600,8=1200,A=2400,C=4800,E=9600,F=19200
                         ;
0A1F  00 01 03 05 07     PRTYCNV     DATA.B     0,1,3,5,7                 ;PARITY
                         ; 0=DISABLED,1=ODD,3=EVEN,5=MARK XMIT/NO RCV,7=SPACE XMIT/NO RCV
                         ;
0A24  49                 HNDSCNV     DATA.B     $49                       ;LINE/CTS/INV
0A25  09                             DATA.B     $09                       ;LINE/CTS/NOT INV
0A26  51                             DATA.B     $51                       ;LINE/DSR/INV
0A27  11                             DATA.B     $11                       ;LINE/DSR/NOT INV
0A28  61                             DATA.B     $61                       ;LINE/DCD/INV
0A29  21                             DATA.B     $21                       ;LINE/DCD/NOT INV
0A2A  02                             DATA.B     $02                       ;XON/XOFF
0A2B  04                             DATA.B     $04                       ;ENQ/ACK
0A2C  80                             DATA.B     $80                       ;ETX/ACK
0A2D  00                             DATA.B     $00                       ;NONE OF THE ABOVE PROTOCOLS
                         ;=====================================================================================;
```

```
                  ;
                  ; Variable data area
                  ;
                  ; Port 0 data area
                  ;
0A2E              Port0Data

                  ; DEFAULT BUFFER  Control Table  - MUST HAVE SAME FIELD FORMAT AS BUFFER CONTROL TABLE
                  ;
      00000000    DEFBWRT     EQU      %-Port0Data
0A2E  0E                      DATA.B   $0E                 ;WRITE BAUD RATE-9600
      00000001    DEFBRD      EQU      %-Port0Data
0A2F  0E                      DATA.B   $0E                 ;READ BAUD RATE-9600
      00000002    DEFPART     EQU      %-Port0Data
0A30  00                      DATA.B   $00                 ;PARITY-DISABLED
      00000003    DEFWRDS     EQU      %-Port0Data
0A31  00                      DATA.B   $00                 ;WORD SIZE = 8 BITS (1=7 BITS)
      00000004    DEFBTWNEA   EQU      %-Port0Data         ;NUMBER OF CHARS BETWEEN
0A32  0050                    DATA.W   80                  ;        ENQ's or ETX's
      00000006    DEFINTRN    EQU      %-Port0Data
0A34  0000                    DATA.W   $0000               ;INTERNAL FLAG--all off
      00000008    DEFPROT     EQU      %-Port0Data
0A36  0902                    DATA.W   $0902               ;PROTOCOL FLAG--Enabled - XON/XOFF
      0000000A    DEFend      EQU      %-Port0Data
      00000005    DEFBCTLN    EQU      (DEFend-DEFBWRT)/2  ;number of words in both tables

                  ; BUFFER CONTROL TABLE
                  ;
      0000000A    BFRCTL      EQU      %-Port0Data         ;Index to Buffer Control Table
      0000000A    BF_WRBD     EQU      %-Port0Data         ;Index to WRITE BAUD RATE
0A38  00                      DATA.B   0                   ;
      0000000B    BF_RDBD     EQU      %-Port0Data         ;Index to READ BAUD RATE
0A39  00                      DATA.B   0                   ;
      0000000C    BF_PART     EQU      %-Port0Data         ;Index to PARITY
0A3A  00                      DATA.B   0                   ;
      0000000D    BF_WRDS     EQU      %-Port0Data         ;Index to WORD SIZE
0A3B  00                      DATA.B   0                   ;
      0000000E    BF_BTWNEA   EQU      %-Port0Data         ;Index to NUMBER OF CHARS BETWEEN
0A3C  0000                    DATA.W   0                   ;        ENQ's or ETX's
      00000010    BF_INTL     EQU      %-Port0Data         ;Index to INTERNAL FLAGS
0A3E  0000                    DATA.W   0                   ;
      00000012    BF_PROF     EQU      %-Port0Data         ;Index to PROTOCOL FLAGS-HANDSHAKE TYPE
0A40  0000                    DATA.W   0                   ;

                  ;           WRITE BUFFER CONTROL TABLE
                  ;
      00000014    WRTCTL      EQU      %-Port0Data         ;Index to WRITE BUFFER CONTROL TABLE
      00000014    WB_FLG1     EQU      %-Port0Data         ;Index to FLAG WORD 1
0A42  0000                    DATA.W   0
      00000016    WB_FLG2     EQU      %-Port0Data         ;Index to FLAG WORD 2
0A44  0000                    DATA.W   0
      00000018    WB_FILLP    EQU      %-Port0Data         ;Index to BUFFER FILL POINTER  rear
0A46  00000000                DATA.L   0
      0000001C    WB_EMPTY    EQU      %-Port0Data         ;Index to BUFFER EMPTY POINTER front
```

```
0A4A 00000000           2324*           DATA.L    0
     00000020           2325* WB_BADR   EQU       %-Port0Data          ,Index to BUFFER ADDRESS
0A4E 00000000           2326*           DATA.L    0
     00000024           2327* WB_SIZE   EQU       %-Port0Data          ,Index to BUFFER SIZE
0A52 0000               2328*           DATA.W    0
     00000026           2329* WB_FREE   EQU       %-Port0Data          ,Index to AMOUNT OF BUFFER FREE SPACE
0A54 0000               2330*           DATA.W    0
     00000028           2331* WB_ABADR  EQU       %-Port0Data          ,Index to ALTERNATE BUFFER ADDRESS
0A56 00000000           2332*           DATA.L    0
     0000002C           2333* WB_ASIZE  EQU       %-Port0Data          ,Index to ALTERNATE BUFFER SIZE
0A5A 0000               2334*           DATA.W    0
     0000002E           2335* WB_BENQ   EQU       %-Port0Data          ,Index to Number of bytes before wait for A
CK
0A5C 0000               2336*           DATA.W    0
                        2337* ,
                        2338* ;         READ BUFFER CONTROL TABLE
                        2339* ;
     00000030           2340* RDCTL     EQU       %-Port0Data          ,Index to READ BUFFER CONTROL TABLE
     00000030           2341* RB_FLG1   EQU       %-Port0Data          ;Index to FLAG WORD 1
0A5E 0000               2342*           DATA.W    0
     00000032           2343* RB_FLG2   EQU       %-Port0Data          ,Index to FLAG WORD 2
0A60 0000               2344*           DATA.W    0
     00000034           2345* RB_FILLP  EQU       %-Port0Data          ;Index to BUFFER FILL POINTER  rear
0A62 00000000           2346*           DATA.L    0
     00000038           2347* RB_EMPTY  EQU       %-Port0Data          ,Index to BUFFER EMPTY POINTER front
0A66 00000000           2348*           DATA.L    0
     0000003C           2349* RB_BADR   EQU       %-Port0Data          ;Index to BUFFER ADDRESS
0A6A 00000000           2350*           DATA.L    0
     00000040           2351* RB_SIZE   EQU       %-Port0Data          ,Index to BUFFER SIZE
0A6E 0000               2352*           DATA.W    0
     00000042           2353* RB_FREE   EQU       %-Port0Data          ,Index to AMOUNT OF BUFFER FREE SPACE
0A70 0000               2354*           DATA.W    0
     00000044           2355* RB_ABADR  EQU       %-Port0Data          ,Index to ALTERNATE BUFFER ADDRESS
0A72 00000000           2356*           DATA.L    0
     00000048           2357* RB_ASIZE  EQU       %-Port0Data          ,Index to ALTERNATE BUFFER SIZE
0A76 0000               2358*           DATA.W    0
     0000004A           2359* RB_HIWA   EQU       %-Port0Data          ,Index to NUMBER OF BYTES IN HI WATER MARK
0A78 0000               2360*           DATA.W    0                    ;number of bytes in buffer when at hi water
mark
     0000004C           2361* RB_LOWA   EQU       %-Port0Data          ,Index to NUMBER OF BYTES IN LOW WATER MARK
0A7A 0000               2362*           DATA.W    0                    ,number of bytes in buffer when at low wate
r mark
                        2363* ,
                        2364* , control character buffer
                        2365* ;
     0000004E           2366* CB_FRONT  EQU       %-Port0Data          ;Index to Ctl buffer Front Pointer
0A7C 00000000           2367*           DATA.L    0
     00000052           2368* CB_REAR   EQU       %-Port0Data          ,Index to Ctl buffer Rear Pointer
0A80 00000000           2369*           DATA.L    0
     00000056           2370* CB_FLAGS  EQU       %-Port0Data          ;Index to Ctl buffer Flags word
0A84 0000               2371*           DATA.W    0
     00000058           2372* CTLBUF    EQU       %-Port0Data          ,Index to Ctl buffer
0A86 00000000           2373*           DATA.L    0
                        2374* ;
                        2375* ,  Read Buffer - 256 bytes
                        2376* ,
     0000005C           2377* RDBUF     EQU       %-Port0Data          ;Index to Read Buffer
```

```
0A8A  00000000          2378*           DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0        ;64
0A8E  00000000
0A92  00000000
0A96  00000000
0A9A  00000000
0A9E  00000000
0AA2  00000000
0AA6  00000000
0AAA  00000000
0AAE  00000000
0AB2  00000000
0AB6  00000000
0ABA  00000000
0ABE  00000000
0AC2  00000000
0AC6  00000000
0ACA  00000000          2379*           DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0        ,128
0ACE  00000000
0AD2  00000000
0AD6  00000000
0ADA  00000000
0ADE  00000000
0AE2  00000000
0AE6  00000000
0AEA  00000000
0AEE  00000000
0AF2  00000000
0AF6  00000000
0AFA  00000000
0AFE  00000000
0B02  00000000
0B06  00000000
0B0A  00000000          2380*           DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0        ,
0B0E  00000000
0B12  00000000
0B16  00000000
0B1A  00000000
0B1E  00000000
0B22  00000000
0B26  00000000
0B2A  00000000
0B2E  00000000
0B32  00000000
0B36  00000000
0B3A  00000000
0B3E  00000000
0B42  00000000
0B46  00000000
0B4A  00000000          2381*           DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0        ,256
0B4E  00000000
0B52  00000000
0B56  00000000
0B5A  00000000
0B5E  00000000
```

```
0B62  00000000
0B66  00000000
0B6A  00000000
0B6E  00000000
0B72  00000000
0B76  00000000
0B7A  00000000
0B7E  00000000
0B82  00000000
0B86  00000000
      0000015C          2382* RBFend      EQU       %-PortOData
      00000100          2383* RBFLEN      EQU       RBFend-RDBUF            ,READ BUFFER LENGTH
                        2384* .
                        2385* ,  Write Buffer - 256 bytes
                        2386* ,
      0000015C          2387* WRTBUF      EQU       %-PortOData             ;Index to Write Buffer
0B8A  00000000          2388*             DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0       ;64
0B8E  00000000
0B92  00000000
0B96  00000000
0B9A  00000000
0B9E  00000000
0BA2  00000000
0BA6  00000000
0BAA  00000000
0BAE  00000000
0BB2  00000000
0BB6  00000000
0BBA  00000000
0BBE  00000000
0BC2  00000000
0BC6  00000000
0BCA  00000000          2389*             DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0       ;128
0BCE  00000000
0BD2  00000000
0BD6  00000000
0BDA  00000000
0BDE  00000000
0BE2  00000000
0BE6  00000000
0BEA  00000000
0BEE  00000000
0BF2  00000000
0BF6  00000000
0BFA  00000000
0BFE  00000000
0C02  00000000
0C06  00000000
0C0A  00000000          2390*             DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0       ;
0C0E  00000000
0C12  00000000
0C16  00000000
0C1A  00000000
0C1E  00000000
```

```
0C22 00000000
0C26 00000000
0C2A 00000000
0C2E 00000000
0C32 00000000
0C36 00000000
0C3A 00000000
0C3E 00000000
0C42 00000000
0C46 00000000
0C4A 00000000          2391*              DATA.L      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0         ,256
0C4E 00000000
0C52 00000000
0C56 00000000
0C5A 00000000
0C5E 00000000
0C62 00000000
0C66 00000000
0C6A 00000000
0C6E 00000000
0C72 00000000
0C76 00000000
0C7A 00000000
0C7E 00000000
0C82 00000000
0C86 00000000
     0000025C          2392* WBFend       EQU        %-PortOData
     00000100          2393* WBFLEN       EQU        WBFend-WRTBUF             ;WRITE BUFFER LENGTH
                       2394* ;
     0000025C          2395* pdlen        EQU        %-PortOData               ;Length of port data area
```

```
                       2397* ;
                       2398* ; Port 1 data area
                       2399* ; same structure as the port 0 data area
                       2400* ;
0C8A                   2401* Port1Data
                       2402*
                       2403* ; DEFAULT BUFFER  Control Table
                       2404*
0C8A  0C               2405*          DATA.B    $0C                   ;WRITE BAUD RATE-4800
0C8B  0C               2406*          DATA.B    $0C                   ;READ BAUD RATE-4800
0C8C  00               2407*          DATA.B    $00                   ;PARITY-DISABLED
0C8D  00               2408*          DATA.B    $00                   ;WORD SIZE = 8 BITS (1=7 BITS)
0C8E  0050             2409*          DATA.W    80                    ;NUMBER OF CHARS BETWEEN ENQ's or ETX's
0C90  0000             2410*          DATA.W    $00                   ;INTERNAL FLAG--all off
0C92  0911             2411*          DATA.W    $0911                 ;PROTOCOL FLAG--Enabled/LINE/DSR/NOT INV
                       2412*
0C94  00 00 00 00      2413*          DATA.B    0,0,0,0               ; buffer control table
0C98  0000  0000  0000 2414*          DATA.W    0,0,0
                       2415*
0C9E  00000000         2416*          DATA.L    0,0,0,0               ; write buffer control table
0CA2  00000000
0CA6  00000000
0CAA  00000000
0CAE  0000  0000  0000 2417*          DATA.W    0,0,0,0,0
0CB4  0000  0000
                       2418*
0CB8  00000000         2419*          DATA.L    0,0,0,0               ; read buffer control table
0CBC  00000000
0CC0  00000000
0CC4  00000000
0CC8  0000  0000  0000 2420*          DATA.W    0,0,0,0,0,0,0,0
0CCE  0000  0000  0000
0CD4  0000  0000
                       2421*
                       2422* ; control character buffer and control variables
                       2423* ;
0CD8  00000000         2424*          DATA.L    0,0
0CDC  00000000
0CE0  0000             2425*          DATA.W    0
0CE2  00000000         2426*          DATA.L    0
                       2427* ;
0CE6  00000000         2428*          DATA.L    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0        ;64
0CEA  00000000
0CEE  00000000
0CF2  00000000
0CF6  00000000
0CFA  00000000
0CFE  00000000
0D02  00000000
0D06  00000000
0D0A  00000000
0D0E  00000000
0D12  00000000
0D16  00000000
```

```
0D1A  00000000
0D1E  00000000
0D22  00000000
0D26  00000000        2429*            DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ;128
0D2A  00000000
0D2E  00000000
0D32  00000000
0D36  00000000
0D3A  00000000
0D3E  00000000
0D42  00000000
0D46  00000000
0D4A  00000000
0D4E  00000000
0D52  00000000
0D56  00000000
0D5A  00000000
0D5E  00000000
0D62  00000000
0D66  00000000        2430*            DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ,
0D6A  00000000
0D6E  00000000
0D72  00000000
0D76  00000000
0D7A  00000000
0D7E  00000000
0D82  00000000
0D86  00000000
0D8A  00000000
0D8E  00000000
0D92  00000000
0D96  00000000
0D9A  00000000
0D9E  00000000
0DA2  00000000
0DA6  00000000        2431*            DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ,256
0DAA  00000000
0DAE  00000000
0DB2  00000000
0DB6  00000000
0DBA  00000000
0DBE  00000000
0DC2  00000000
0DC6  00000000
0DCA  00000000
0DCE  00000000
0DD2  00000000
0DD6  00000000
0DDA  00000000
0DDE  00000000
0DE2  00000000
                      2432*
                      2433* ; write buffer
                      2434* ;
```

```
0DE6  00000000          2435*          DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ;64
0DEA  00000000
0DEE  00000000
0DF2  00000000
0DF6  00000000
0DFA  00000000
0DFE  00000000
0E02  00000000
0E06  00000000
0E0A  00000000
0E0E  00000000
0E12  00000000
0E16  00000000
0E1A  00000000
0E1E  00000000
0E22  00000000
0E26  00000000          2436*          DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ;128
0E2A  00000000
0E2E  00000000
0E32  00000000
0E36  00000000
0E3A  00000000
0E3E  00000000
0E42  00000000
0E46  00000000
0E4A  00000000
0E4E  00000000
0E52  00000000
0E56  00000000
0E5A  00000000
0E5E  00000000
0E62  00000000
0E66  00000000          2437*          DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ;
0E6A  00000000
0E6E  00000000
0E72  00000000
0E76  00000000
0E7A  00000000
0E7E  00000000
0E82  00000000
0E86  00000000
0E8A  00000000
0E8E  00000000
0E92  00000000
0E96  00000000
0E9A  00000000
0E9E  00000000
0EA2  00000000
0EA6  00000000          2438*          DATA.L     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0          ;256
0EAA  00000000
0EAE  00000000
0EB2  00000000
0EB6  00000000
0EBA  00000000
```

```
0EBE  00000000
0EC2  00000000
0EC6  00000000
0ECA  00000000
0ECE  00000000
0ED2  00000000
0ED6  00000000
0EDA  00000000
0EDE  00000000
0EE2  00000000
```

```
                       ;
                       ;Common area
                       ;
                       ; Flags
                       ;
0EE6  0000             CMNFLGS     DATA.W     0

                       ; Unit Numbers for the ports
                       ;
0EE8  0000             UnitP0      DATA.W     0
0EEA  0000             UnitP1      DATA.W     0

                       ; Save of DataCom Ctrl interrupt vector
                       ;
0EEC  00000000         SaveLvl1    DATA.L     0

      00000000+                    END        COMDRV

 ACK       00000006  CKBUFERR   0002C6+  COMREX      0001F4+  D2_BENQ   00000012  DEFINTRN 00000006
 ACTIVE    00000006  CKPORT     0001D6+  COMST       000706+  D2_BFCTR  00000006  DEFPART  00000002
 ALTBF_R1  00000002  CKRDERR    0001CA+  COMTBL      000056+  D2_CHARS  00000004  DEFPROT  00000008
 ALTBF_W1  00000002  CKWRTP     0002D2+  COMUNMT     0006D2+  D2_FREER  00000003  DEFWRDS  00000003
 AULF_W2   00000004  CLNENBL    00061C+  COMWEX      0002F8+  D2_FREEW  00000000  DISINT1  00002100
 BAUDCNV     000A18+ CLNEXIT    000620+  COMWR       0002B6+  D2_HANDS  00000005  DISINT2  00002200
 BFRCTL    0000000A  CLRBUSY    0006B8+  CPREXIT     000288+  D2_INBRD  0000000C  DISINT4  00002400
 BF_BTWNE  0000000E  CLRCMD   00000003   CPRION      000284+  D2_INBWT  0000000E  DISINTS    000096+
 BF_INTL   00000010  CLRD3D2  000000F3   CR        0000000D   D2_OUTRD  00000008  DISRCVIN   000936+
 BF_PART   0000000C  CLRSC    00000006   CRBYEXIT    0004BC+  D2_OUTWT  0000000D  DITEXIT    0000BA+
 BF_PROF   00000012  CMDRC    00000009   CR_BDCLK  00000010   D2_PARTY  00000002  DITISP0    0000A4+
 BF_RDBD   0000000B  CMDREGI  00000005   CR_EXTCL  00000000   D2_RBCHR  00000010  DODISINT   0008CC+
 BF_WRBD   0000000A  CMDRWC   00000005   CR_STPB   00000080   D2_RDALT  00000013  DOENBINT   0008F2+
 BF_WRDS   0000000D  CMNFLGS    000EE6+  CR_WRDL5  00000060   D2_REAHI  00000009  DRIEXIT    000950+
 BITD0     00000000  CM_DISP  00000000   CR_WRDL6  00000040   D2_REALO  0000000A  DSRLIN   00000004
 BITD1     00000001  CM_DTRL  00000001   CR_WRDL7  00000020   D2_RESTS  00000007  EITEXIT    0000C4+
 BITD2     00000002  CM_ECHO  00000010   CR_WRDL8  00000000   D2_WBCHR  0000000F  EMPT_CB  00000001
 BITD3     00000003  CM_EPBT  00000060   CSATTR1   00000010   D2_WRSTS  00000008  EMPT_R2  00000001
 BITD4     00000004  CM_IRQD  00000002   CSATTR2   00000011   D2_WTALT  00000014  EMPT_W2  00000001
 BITD5     00000005  CM_MPBD  000000A0   CSBPCH    00000004   DATAREG   00000001  ENBINTS    0000BC+
 BITD6     00000006  CM_OPBT  00000020   CSFRSTCH  00000008   DC0INT      00039C+ ENBRCVIN   000928+
 BITD7     00000007  CM_SPBD  000000E0   CSLASTCH  0000000A   DC1INT      0003AA+ ENQ      00000005
 BRBCHR      00095E+ CM_TDBRK 0000000C   CSLPCH    00000004   DC1OFF    00000020  ENQACK   00000002
 BUSYCMD   00000004  CM_TDHI  00000000   CSMASK    0000000C   DCDLIN    00000005  ENQFLG   00000004
 BUSY_R1   00000000  CM_TDLO  00000008   CSTBLLOC  00000000   DCICOMN     0003B6+ ERICHKPR   000932+
 BUSY_W1   00000008  CM_TELO  00000004   CSTERR      00071C+  DCIEXIT     0003D0+ ERR_R1   00000001
 BWBCHR      000952+ CNTCHARS   000596+  CSTTBL      000720+  DCIPX       0003CC+ ERR_W1   00000001
 CARRYST   00000001  CNTENQ     0005B0+  CTLBUF    00000058   DCIRCV      0003BE+ ETX      00000003
 CBSEXIT     0006C2+ CNTEXIT    0005BE+  CTLRC     00000010   DCIXMIT     0003C6+ ETXACK   00000007
 CB_FLAGS  00000056  COM001     000020+  CTLREGI   00000007   DCTLINT     0005C0+ FINDLIN    000622+
 CB_FRONT  0000004E  COMBSY     0006C4+  CTSLIN    00000003   DDRA      00030F67  FLNEXIT    00063E+
 CB_REAR   00000052  COMCLR     000654+  CUMDIFF     000700+  DEFBCTLN  00000005  FLNGOT     000636+
 CHKLINES    0005E2+ *COMDRV    000000+  CUMISP0     0006F4+  DEFBRD    00000001  FLNLOOK    000626+
 CHKPROTO    000244+ COMINST    000064+  CURSON    00000002   DEFBTWNE  00000004  FULL_CB  00000000
 CHKRCVBU    0004AE+ COMISP0    000040+  D2_ATLF   00000011   DEFBWRT   00000000  FULL_P2  00000003
 CINBUFCT    000072+ COMRD      0001BA+  D2_BAUDS  00000001   DEFEND    0000000A  FULL_R2  00000000
```

```
FULL_W2  00000000   IOENOTIM 0000002A   OFF        00000000  RCHKALTB   00028A+  SHDEXIT     0007D6+
GABFEXIT   0009B0+  IOENOTRN 00000015   ON         00000001  RDABEXIT   0009A4+  SHDZERO     0007D2+
GBSISP0    000120+  IOEORDSB 0000003D   ORA        00030F63  RDABOK     000984+  SNDLF_W2 00000003
GCLEXIT    00039A+  IOEOWDSB 0000003F   OTWTOFF      00090C+ RDABSWTC   0009A2+  SNXCNT      00055C+
GETALTBU   0009A8+  IOEPADER 00000044   OUTC_R1    00000006  RDABSZ     00097C+  SNXEXIT     00056A+
GETBASE    000110+  IOEPRMLN 00000037   OUTC_W1    00000006  RDBUF    0000005C   SNXNOTMT    000546+
GETCTL     000384+  IOERSZER 00000042   OUTE_R1    00000004  RDCTL    00000030   SNXNOWRP    000528+
GETVAL     0008C0+  IOETBLFL 00000033   OUTE_W1    00000004  RDSTEXIT   000836+  SNXSZERR    00053E+
GORCVBUS   0004BE+  IOETBLID 00000032   PCBEXIT      0004AC+ RDSTNONE   000830+  STACSLT  00000004
GOUNBUSY   000272+  IOETBLIU 00000034   PCBNOTFL     000494+ READCMD  00000001   STACSRV  00000006
GRAPHIC  00000001   IOETIMOT 00000016   PCBNOWRP     000488+ READONE    0001EA+  STALSLT  00000008
GRBSEXIT   0004D4+  IOEUARTE 00000043   PDCCONTL     000420+ REREAD     0001BE+  STALSRV  0000000A
HELPRD     000238+  IOEUIOPM 00000036   PDCLACK      00044A+ REWRITE    00029A+  STATEMSK 00002000
HILOMSK  000000F0   IOEWNDBE 00000021   PDCLCHKX     000456+ RSTUART    000792+  STATOLF     0007E4+
HMLEN    00000017   IOEWNDCS 00000022   PDCLDIDI     000468+ RSTUART1   000794+  STATRI   00000003
HNDSCNV    000A24+  IOEWNDDC 00000023   PDCLENQ      00043A+ SAVELVL1   000EEC+  STBAUD      000760+
INIRDBF    000176+  IOEWNDDS 00000024   PDCLEXIT     00046A+ SAVEUNIT   0000C6+  STBENQ      0007EC+
INITBUFS   000666+  IOEWNDFN 00000020   PDCLSEND     000444+ SAVPARM    00078E+  STBFERR     000980+
INITCTLB   0001A4+  IOEWNDIW 00000025   PDCLXOFF     000462+ SBFSISP0   0008AC+  STBFSTS     000884+
INIWRBF    000152+  IOEWNDWN 00000027   PDCLXON      00044A+ SCBOOTDV 00000036   STBTSLT  00000000
INPC_R1  00000007   IOEWNDWR 00000026   PDLEN      0000025C  SCBOOTNM 0000002E   STBTSRV  00000002
INPC_W1  00000007   IOEWSZER 00000041   PORT0DAT     000A2E+ SCCODEJT 00000022   STCALLDI    00074A+
INPE_R1  00000005   IOOK     00000000   PORT1DAT     000C8A+ SCCURRK  00000048   STCALLEN    000754+
INPE_W1  00000005   LF       0000000A   PORTFLG    00000000  SCCURRW  00000044   STHNDSK     00079E+
INRDENB    0008F0+  LFSPRSFL 0000000C   PRCERROR     000410+ SCDEVTAB 00000014   STINFO   0000000C
INSMOD   00000002   LINE     00000000   PRCEXIT      00041E+ SCDIRNAM 00000018   STINFOL  00000004
INSTCMD  00000000   LOST_R2  00000002   PRCLSTDT     000418+ SCFREEHP 00000004   STINRD      0008E0+
INT1     00000100   LOST_W2  00000002   PRCNOCTL     0003FC+ SCIORSLT 00000000   STINWT      000912+
INT2     00000200   MAKEBOOL   000838+  PRCVCHAR     0003D6+ SCJTABLE 00000008   STNDRV   00000002
INT4     00000400   MAXBAUD  00000006   PRNDERR      000054+ SCMEMMAP 00000032   STNMBR   00000000
INTMSK   00000700   MAXDTCM  00000001   PROT_P2    00000000  SCNUMPRO 00000028   STOPXMIT    00035C+
INVBUSY  00000006   MAXHNDS  00000009   PRTYCNV      000A1F+ SCNXTPRO 00000026   STOUTRD     0008D0+
INVCMD   00000003   MAXPRTY  00000004   PRXEXIT      0004FE+ SCPROCNO 00000002   STOUTWT     0008F6+
INVCURS  00000003   MAXRHI   00000085   PRXGETCT     0004FC+ SCPROTBL 0000002A   STPRITY     000770+
INVFNC   00000038   MAXRLO   00000050   PRXMIT       0004D6+ SCROOTW  00000040   STRBUF      000758+
INVPRM   00000036   MAXWHI   00000085   PRXOFF       0004F2+ SCSLTTBL 0000003C   STRDALTB    00096A+
INVRSE   00000000   MAXWLO   00000050   PRXSEND      0004F8+ SCSUSINH 0000005A   STRDHI      0007D8+
INVTBLID 00000032   MAXWRDS  00000001   PSYSCOM    00000180  SCSUSREQ 0000005C   STRDLO      0007DE+
IODDRA   00000080   MMBTBLK  0000001A   PUTCHRBF     00046C+ SCSYSIN  00000010   STRDSTS     0007F2+
IOEBSZER 00000040   MMBTDEV  00000012   PUTCTL       000372+ SCSYSOUT 0000000C   STRTXMIT    000358+
IOECLKMF 00000039   MMBTDRV  00000018   RBFEND     0000015C  SCTODAY  00000020   STSCMD   00000005
IOEFNCCD 00000038   MMBTSLT  00000014   RBFLEN     00000100  SCUSERID 0000004C   STTYPE   00000001
IOEINVDE 00000002   MMBTSRV  00000016   RB_ABADR   00000044  SCUTABLE 0000001C   STUPEXIT    000A16+
IOEIOREQ 00000003   MMBTSW   00000010   RB_ASIZE   00000048  SCVRSDAT 00000052   STUPGO      0009FC+
IOEIRDSB 0000003C   MMHICOD  0000000C   RB_BADR    0000003C  SCVRSNBR 0000004E   STWBUF      00074E+
IOEIWDSB 0000003E   MMHIDTA  00000004   RB_EMPTY   00000038  SCWNDTBL 00000056   STWRDSZ     000780+
IOEKYBTE 00000035   MMLOCOD  00000008   RB_FILLP   00000034  SENDCTL    000500+  STWTALTB    0009B2+
IOENFDRV 0000002D   MMLODTA  00000000   RB_FLG1    00000030  SENDNEXT   00050A+  STWTSTS     000842+
IOENOBUF 00000017   MODM_P2  00000001   RB_FLG2    00000032  SETERR     00079A+  SUSPEND  00000007
IOENODSP 00000028   NHIRA    00030F7F   RB_FREE    00000042  SETUART    0000E4+  SVBLKIO  0000002C
IOENODTC 0000002E   NMOD_P2  00000002   RB_HIWA    0000004A  SETUPRB    0009F4+  SVCDOPI     000148+
IOENOKYB 00000029   NOAUTOLF 00000004   RB_LOWA    0000004C  SETUPWB    0009EC+  SVCEXIT     000150+
IOENOOMN 0000002B   NOSCROLL 00000005   RB_SIZE    00000040  SETVECS    000122+  SVCLI    0000007C
IOENOPRT 0000002C   NULL     00000000   RCABEXIT     0002B4+ SHDCHKEA   0007C2+  SVCLOSE  00000020
```

```
SVCRKPTH 00000060   SVWRCHAR 00000024   UPCISON     000354+ WB_ABADR 00000028   WRGRORGY 0000001E
SVCSAME     000138+ SXTGETB     00035E+ UPCNOTCR    000342+ WB_ASIZE 0000002C   WRHOMEOF 0000000C
SVDELENT 00000090   SYSBYTES 00000186   UPCNOTFL    000326+ WB_BADR  00000020   WRHOMEPT 00000004
SVD5F    00000038   SYSKYBDF 00000184   UPCNOWRP    00031A+ WB_BENQ  0000002E   WRLENGTH 00000030
SVDSP4   0000006C   SYSWIN   00000005   UPRMSK   0000A000   WB_EMPTY 0000001C   WRLNGTHX 00000012
SVFLPDIR 00000088   S_DCD    00000005   UPUTCHR     0002FA+ WB_FILLP 00000018   WRLNGTHY 00000014
SVGET    00000014   S_DSR    00000006   UTBLF    00000006   WB_FLG1  00000014   WRPROB      0002F8+
SVGETDIR 00000048   S_ERRBIT 00000007   UTBLK    0000001C   WB_FLG2  00000016   WRRCDLEN 00000023
SVGETVNM 00000080   S_FRAME  00000001   UTDID    00000008   WB_FREE  00000026   WRSTATE  00000022
SVINIT   00000018   S_IRQ    00000007   UTDRV    00000016   WB_SIZE  00000024   WRTANLF     0002F4+
SVMARK   0000003C   S_OVRN   00000002   UTFLP    00000018   WCABEXIT    000594+ WRTBUF   0000015C
SVMAVAIL 00000044   S_PARI   00000000   UTIODRV  00000002   WCHKALTB    00056C+ WRTCTL   00000014
SVNEW    00000034   S_RCVF   00000003   UTLEN    00000020   WRAPON   00000004   WRTONE      0002E6+
SVNEW4   00000068   S_WRTE   00000004   UTMTD    00000007   WRATTR1  00000020   WRWWSPTR 0000002C
SVOPEN   0000001C   TBLSTATE 00000014   UTRO     0000001A   WRATTR2  00000021   WTABEXIT    0009E8+
SVPUT    00000010   THERTE      000704+ UTSIZ    00000010   WRATTR3  00000024   WTABOK      0009C8+
SVPUTDIR 00000094   TRACEMSK 00008000   UTSLT    00000014   WRBASEX  0000000E   WTABSWTC    0009E6+
SVRDCHAR 00000028   TRUE     00000001   UTSPT    00000018   WRBASEY  00000010   WTABSZ      0009C4+
SVRLEASE 00000040   TSTLINE     000640+ UTSRV    00000015   WRBITOFS 0000001A   WTSTEXIT    000882+
SVSCHDIR 0000008C   TURNOFF  00000002   UTTPS    00000019   WRCHARPT 00000000   WTSTNONE    00087C+
SVSEEK   00000030   UARTDC0  00030F20   UTTYP    00000017   WRCMD    00000002   XMITDIS  00000008
SVUBUSY  0000000C   UGCNOTMT    00022E+ VEC1     00000064   WRCURADR 00000008   XMITENB  00000004
SVUCLEAR 00000008   UGCNOWRP    000216+ VEC2     00000068   WRCURSX  00000016   XOFF     00000013
SVUINSTL 00000098   UGETCHR     0001F6+ VEC4     00000070   WRCURSY  00000018   XON      00000011
SVUISP0     0000E0+ UNDSCR   00000001   VERT     00000000   WRFILL1  00000025   XONXOFF  00000001
SVUREAD  00000004   UNITP0      000EE8+ VIDDEFLT 00000003   WRFILL2  00000026   XXX010      000009+
SVUSTAT  00000064   UNITP1      000EEA+ VIDSET   00000007   WRFILL3  00000027
SVUWRITE 00000000   UNMCMD   00000006   WBFEND   0000025C   WRFILL4  00000028
SVVALDIR 00000084   UPCISALF    00033C+ WBFLEN   00000100   WRGRORGX 0000001C

0 errors.  2456 lines.  File DRV.DTACOM.TEXT
```

NOTE

THE FOLLOWING EXAMPLE IS A LISTING OF THE PRINT WINDOW PROGRAM USED FOR DOING A SCREEN DUMP FROM A TEMPORARY WINDOW.

```
{ PRTWND.TEXT -------------------------------------------------------------}
{
{        PRTWND -- Print Current Window
{
{        (c) Copyright 1982 Corvus Systems, Inc.
{                               San Jose, California
{
{        All Rights Reserved
{
{        v 1.0  07-01-82  KML  Original program for MX100 printer
{        v 1.1  10-01-82  LEF  Added IDS460 printer support
{
{--------------------------------------------------------------------------}

PROGRAM prtwnd;

USES {$U /CCUTIL/CCLIB} CCdefn, CCwndIO;

CONST esc = 27;

TYPE  prtrid = (NONE, IDS, MX100);

VAR   prtype: prtrid;
      prtr:   integer;
      disp:   integer;
      i,argn: integer;
      curWnd: pWndRcd;
      pDev:   pString80;

{- CCLIB external definitions     }

FUNCTION  pOScurWnd: pWndRcd;    { get kybd record pointer } EXTERNAL;
FUNCTION  pOSdevNam (n: integer): pString80;               EXTERNAL;
FUNCTION  OSdispDv: integer;     { get display unit nmbr   } EXTERNAL;

PROCEDURE Rbytes (x,y,count: integer; pBuff: pBytes);
      const RDBYTES = 7;
      var wbuf: record
                bytecount: integer;
                buffptr: pBytes;
                end;
      begin
      if y < 0 then begin
          pBuff^[0] := 0; exit (Rbytes); end;
      write ('\1Bo', chr(x div 256), chr(x mod 256),
                     chr(y div 256), chr(y mod 256), chr(2));
      wbuf.bytecount := count;
      wbuf.buffptr   := pBuff;
      unitstatus (disp,wbuf,RDBYTES);
      end;

{$P}
```

```
PROCEDURE spit (b: byte);
      begin unitwrite (6, b, 1, 0, 1); end;

PROCEDURE doit;
      var i,j,x,y: integer; b: byte;
          cell: array [0..6] of byte;

      FUNCTION bit (i,j: integer): integer;
          var b: integer;
          begin
          b := cell[i]; bit := 0;
          if b < 0 then b := b+256;
          case j of
              0: if odd (b div 128) then bit := 1;
              1: if odd (b div  64) then bit := 1;
              2: if odd (b div  32) then bit := 1;
              3: if odd (b div  16) then bit := 1;
              4: if odd (b div   8) then bit := 1;
              5: if odd (b div   4) then bit := 1;
              6: if odd (b div   2) then bit := 1;
              7: if odd (b)          then bit := 1;
            end;
          end;

      begin
      write ('\1Bb'); { CURSOR OFF }
      with curWnd^ do begin

{ SET LINE SPACING TO 8 DOTS }

          if prtype = MX100 then begin
              spit (esc); spit (ord('A')); spit (8); end;
          if prtype = IDS then begin
              spit (3); end;

{ PRINT LEFT WINDOW BORDER }

          if prtype = MX100 then begin
              spit (esc); spit (ord('K'));
              spit ((lngthy+2) mod 256);
              spit ((lngthy+2) div 256);
              for y := 0 to lngthy+1 do spit (1);
              end;

{$P}
```

```
{ PRINT WINDOW }

          if prtype = MX100 then
              for x := 0 to lngthx div 8 do begin
                  spit (13); spit (10);
                  spit (esc); spit (ord('K'));
                  spit ((lngthy+2) mod 256);
                  spit ((lngthy+2) div 256);
                  spit (-1); { BOTTOM WINDOW BORDER }
                  for y := 0 to lngthy-1 do begin
                      Rbytes (x*8, y, 1, @b);
                      spit (b);
                      end;
                  spit (-1); { TOP WINDOW BORDER }
                  end;
          if prtype = IDS then begin
              y := lngthy-1;
              repeat
                  for x := 0 to lngthx div 8 do begin
                      for i := 0 to 6 do
                          Rbytes (x*8, y-i, 1, @cell[i]);
                      for j := 0 to 7 do begin
                          b := bit (6,j) * 64 +
                               bit (5,j) * 32 +
                               bit (4,j) * 16 +
                               bit (3,j) *  8 +
                               bit (2,j) *  4 +
                               bit (1,j) *  2 +
                               bit (0,j);
                          if b = 3 then spit (3);
                          spit (b);
                          end;
                      end;
                  spit (3); spit (14);
                  y := y-7;
                  until y < 0;
              end;

{ PRINT RIGHT WINDOW BORDER }

          if prtype = MX100 then begin
              spit (13); spit (10);
              spit (esc); spit (ord('K'));
              spit ((lngthy+2) mod 256);
              spit ((lngthy+2) div 256);
              for y := 0 to lngthy+1 do spit (-128);
              end;

{$P}
```

```
{ NORMALIZE PRINTER }

          if prtype = MX100 then begin
              spit (13); spit (10);
              spit (esc); spit (ord('2')); spit (12);
              end;
          if prtype = IDS then begin
              spit (3); spit (2); end;
          end;
      write ('\1Bo'); { CURSOR ON }
      end;


      begin
      CCwndIOinit;
      prtr := 6;
      disp := OSdispDv;
      ourWnd := pOSourWnd;
      pDev := pOSdevNam (prtr);
      if pDev^ <> 'PRINTER' then begin
          writeln ('Printer driver not loaded ....',chr(7)),
          exit (prtwnd);
          end;
      prtype := MX100;
      if argc <> 0 then begin
          prtype := NONE;
          for argn := 1 to argc do begin
              for i := 1 to length(argv[argn]^) do
                  if argv[argn]^[i] in ['a'..'z'] then
                     argv[argn]^[i] := chr(ord(argv[argn]^[i])
                                           -ord('a')+ord('A'));
              if argv[argn]^ = 'IDS'   then prtype := IDS;
              if argv[argn]^ = 'MX100' then prtype := MX100;
              end;
          end;
      if prtype <> NONE
          then doit
          else writeln ('Invalid printer type specified ....',chr(7));
      end.



```

```
VSIXRF -- Cross Reference Listing                          November 20, 1982
File ID: PRTWND.TEXT                                                  Page 5

0                43   44   54   58   63   64   66   94  101  107  116
                117  119  126  133  143  171
1                54   66   67   68   69   70   71   72   73   94  106
                107  108  111  114  118  125  143  173  174
10              102  139  150
12              151
128              66  143
13              102  139  150
14              131
16               69  122
2                46   68   72   92   93  104  105  124  125  141  142
                154
256              45   46   64   92   93  104  105  141  142
27               19
3                69   86  123  127  131  154
32               68  121
4                70   71  122  124
5                71  121
6                54   58   72  117  120  162
64               67  120
7                37   73  119  132  167  184
8                70   84  101  108  116  118  123
ARGC            171  173
ARGN             26  173  174  175  176  178  179
ARGV            174  175  176  178  179
B                53   54   57   61   63   64   66   67   68   69   70
                 71   72   73  108  109  120  127  128
BIT              60   63   66   67   68   69   70   71   72   73  120
                121  122  123  124  125  126
BUFFPTR          40   48
BYTE             53   57   58
BYTECOUNT        39   47
CCDEFN           17
CCWNDIO          17
CCWNDIOINI      161
CELL             58   63  118
CHR              45   46  167  176  184
COUNT            36   47
CURWND           27   79  164
DISP             25   49  163
DIV              45   46   66   67   68   69   70   71   72   93  101
                105  116  142
DOIT             56  183
ESC              19   84   91  103  140  151
I                26   57   60   63  117  118  174  175  176
IDS              21   85  113  153  178
J                57   60   65  119  120  121  122  123  124  125  126
LENGTH          174
LNGTHX          101  116
LNGTHY           92   93   94  104  105  107  114  141  142  143
MOD              45   46   92  104  141
MX100            21   83   90  100  138  149  170  179
N                33
NONE             21  172  182
```

```
VSIXRF -- Cross Reference Listing                                    November 20, 1982
File ID: PRTWND.TEXT                                                            Page 6

ODD            66    67    68    69    70    71    72    73
ORD            84    91   103   140   151   176   177
OSDISPDV       34   163
PBUFF          36    44    48
PBYTES         36    40
PDEV           28   165   166
POSCURWND      32   164
POSDEVNAM      33   165
PRTR           24   162   165
PRTRID         21    23
PRTWND         15   168
PRTYPE         23    83    85    90   100   113   138   149   153   170   172
              178   179   182
PSTRING80      28    33
PWNDRCD        27    32
RBYTES         36    44   108   118
RDBYTES        37    49
SPIT           53    84    86    91    92    93    94   102   103   104   105
              106   109   111   127   128   131   139   140   141   142   143
              150   151   154
UNITSTATUS     49
UNITWRITE      54
WBUF           38    47    48    49
X              36    45    57   101   108   116   118
Y              34    43    46    57    94   107   108   114   118   132   133
              143
```

NOTE

THE FOLLOWING PAGES CONTAIN THE CORVUS CONCEPT KEYBOARD TRANSLATION TABLES.

Table of Contents
KEYBOARD TRANSLATION TABLES

1.0 Overview

2.0 The Keyboard and the Keycodes

3.0 Translation Tables
- 3.1 SHIFT TABLE (STABLE)
- 3.2 REQULAR TABLE (RLTABLE)
- 3.3 ESCAPE # SEQUENCE TABLE (ETABLE)
- 3.4 STANDARD MULTIPLE CHARACTER SEQUENCE TABLE (SMTABLE)
- 3.5 CAPS LOCK & QUALIFIER FLAQ TABLE (CQTABLE)
- 3.6 RELEASE TABLE (RTABLE)
- 3.7 BREAK KEY CODE TABLE (BKEYCOD)

4.0 Translation Table examples
- 4.1 Alphabetic character example
- 4.2 Standard character example
- 4.3 Escape # character example

5.0 Default Keyboard Translation Table

6.0 Program CSK.REV4.TEXT listing

7.0 Program CSK.DANSK.TEXT listing

8.0 Program CSK.GRMN.TEXT listing

1.0 Overview

This document describes the Keyboard Translation Tables and how to build them. These tables are used by the keyboard driver to generate the character sequences corresponding to the key pressed by the user. If a different set of key caps are used or a different set of character codes are desired then new Translation Tables must be built and loaded into the system. This document describes how to perform those operations.

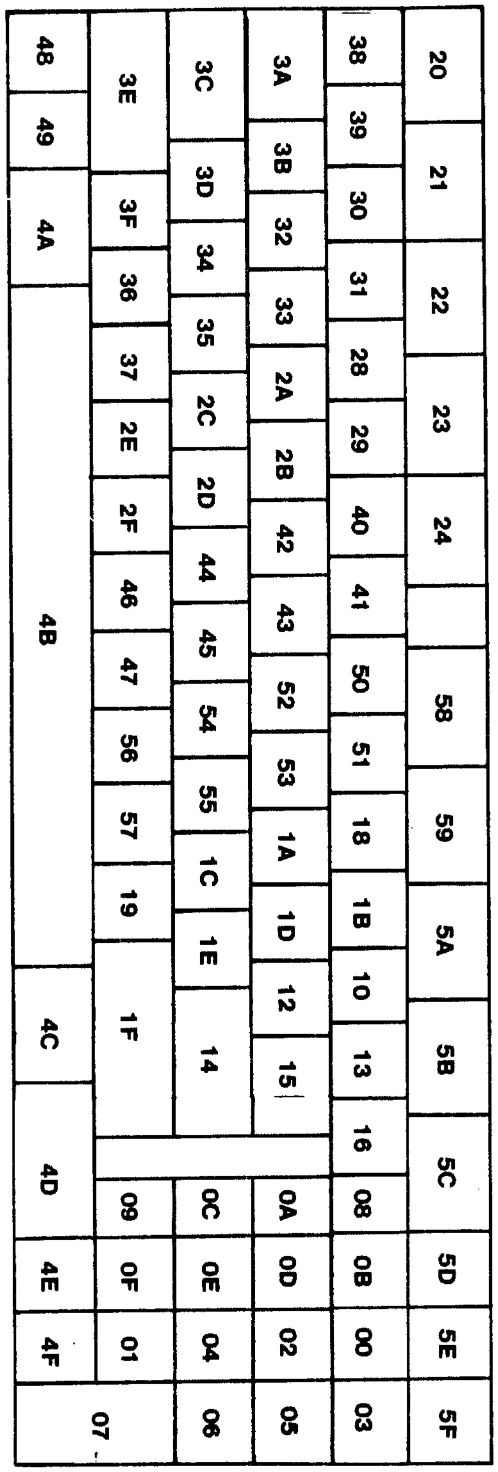
2.0 The Keyboard and Keycodes

The keyboard is connected to the computer by a transmission line. Through the line, the keyboard sends keycodes describing which key has been pressed or released. These keycodes in conjunction with the Translation Tables are used to generate the character sequences produced by the keyboard driver. Some keys, like the Shift key, affect which characters are generated when other keys are pressed. Some keys cause character sequences to be generated. What happens when a key is pressed or release is determined by the Translation Tables.

Keycodes are 8 bits of data, a byte, sent by the keyboard to inform as to which key has been affected and whether it has been pressed (closure) or released. Every key on the Concept keyboard generates 2 keycodes, which differ only by the most significant bit (MSB) of the keycode byte. If the MSB is set (1) then it is the closure. If the MSB is clear (0) then it is the release. The actual character sequence used for a key, whether pressed or released, is determined by decoding the keycodes using the Translation Tables. The keyboard was designed to generate keycodes instead of character sequences which makes the keyboard flexible. By changing the Translation Tables, one can alter the keyboard character set.

In order to build the Translation Tables a Keycode map is needed. This map shows the keycode values for every key on the keyboard. Figure 1 is a Keycode Map for the current keyboard (Version 04, Selectric (R) style keyboard). Normally, the key caps show which character is generated for each keycode transmitted to the keyboard driver. Figure 2 is a key cap map for this same keyboard.

Version 04 keyboard key caps have either 1 or 2 symbols on them. A single symbol key cap specifies that the character is the same when it is either shifted or unshifted, except for the alphabet characters which get lower case if unshifted. Key caps with two symbols have the character for the lower symbol when unshifted and the character for the upper symbol when shifted.


20 21 22 23 24 58 59 5A 5B 5C 5D 5E 5F
38 39 30 31 28 29 40 41 50 51 18 1B 10 13 16 08 0B 00 03
3A 3B 32 33 2A 2B 42 43 52 53 1A 1D 12 15 0A 0D 02 05
3C 3D 34 35 2C 2D 44 45 54 55 1C 1E 14 0C 0E 04 06
3E 3F 36 37 2E 2F 46 47 56 57 19 1F 09 0F 01 07
48 49 4A 4B 4C 4D 4E 4F

Keycode Map (release code)

Figure 1

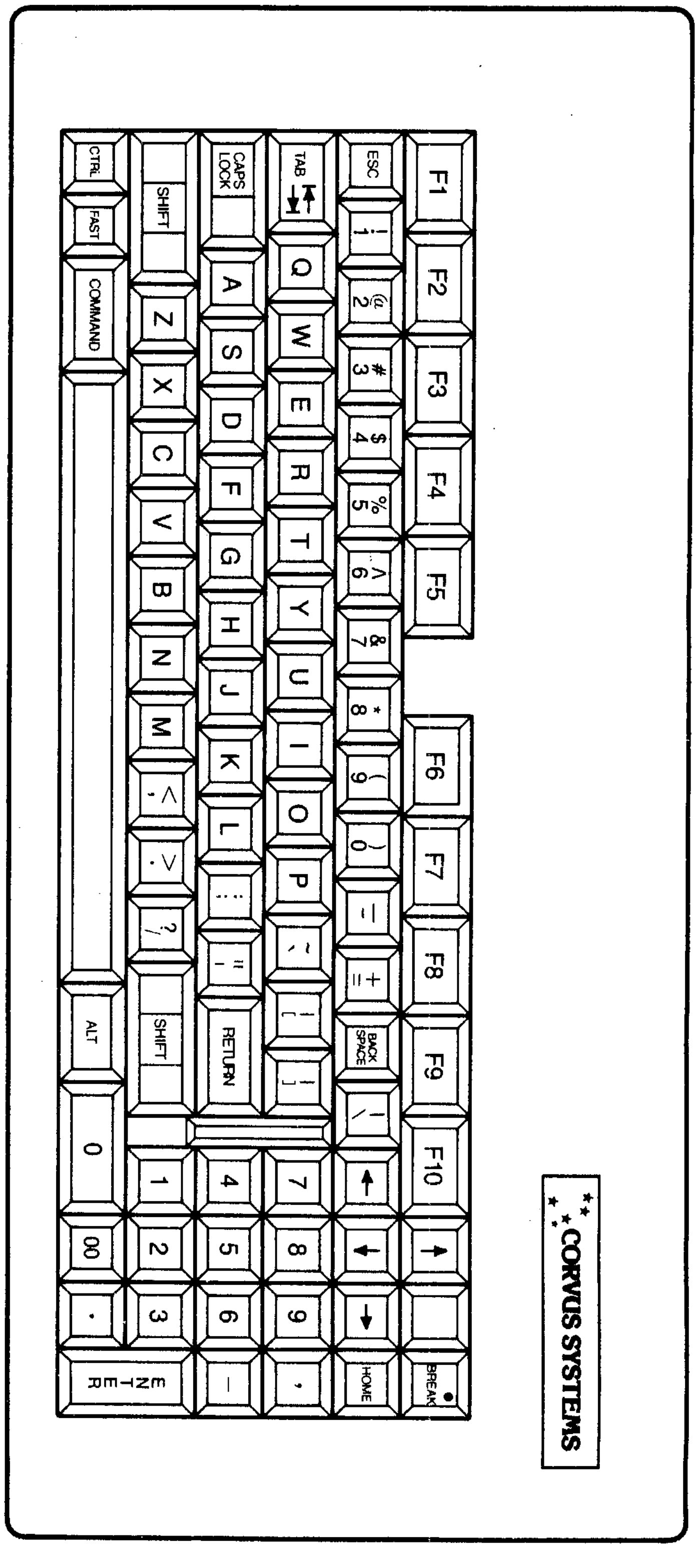

F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
ESC
TAB
CAPS LOCK
SHIFT
CTRL
FAST
COMMAND
ALT
RETURN
BACK SPACE
HOME
BREAK
ENTER
CORVUS SYSTEMS

Figure 2

3.0 Translation Tables

The Translation Tables must be defined in an assembly language program, like the program CSK.REV4.TEXT listed in section 6.0. This program is actually a group of Tables. The first Table is TRANTBL which points to seven the Translation Tables.

The seven entries in this table point to the Translation Tables in the following order:

1) SHIFT_TABLE (STABLE)
2) REGULAR_TABLE (RLTABLE)
3) ESCAPE # SEQUENCE TABLE (ETABLE)
4) STANDARD MULTIPLE CHARACTER SEQUENCE TABLE (SMTABLE)
5) CAPS_QUALIFIER FLAG TABLE (CQTABLE)
6) RELEASE TABLE (RLTABLE)
7) BREAK KEYCODE TABLE (BKEYCOD)

These entries must be in the above order.

3.1 SHIFT TABLE (STABLE)

This table contains one byte for each keycode $00 - $5F. The byte is normally the character code for the specified keycode when the SHIFT key is depressed. Four special byte values are used:

9E - use STANDARD MULTIPLE CHARACTER SEQUENCE TABLE (SMTABLE).
9F - use CAPS_QUALIFIER FLAG TABLE (CQTABLE).
9D - use ESCAPE # SEQUENCE TABLE (ETABLE).
00 - no character for this keycode.

3.2 REGULAR TABLE (RLTABLE)

This table contains one byte for each keycode $00 - $5F. The byte is normally the charater code for the specified keycode when the SHIFT key is not depressed. Four special byte values are used:

9E - use STANDARD MULTIPLE CHARACTER SEQUENCE TABLE (SMTABLE).
9F - use CAPS_QUALIFIER FLAG TABLE (CQTABLE).
9D - use ESCAPE # SEQUENCE TABLE (ETABLE).
00 - no character for this keycode.

3.3 ESCAPE # SEQUENCE TABLE (ETABLE)

This table is used when a table code of $9D is found in key closure or a table code of $9D is found in key SHIFT TABLE

(STABLE) or the REGULAR TABLE (RLTABLE). It specifies a key which has an ESC # character sequence. Each keycode may have a different character based on the state of the two qualifier keys (SHIFT and COMMAND).

Each table entry has the form (entry length = 10 bytes) :

1) Keycode (1 byte).
2) filler byte : its value is 0 (1 byte).
3) UnSHIFTed & UnCOMMANDed (2 bytes).
4) SHIFT only (2 bytes).
5) COMMAND only (2 bytes).
6) COMMAND & SHIFT together (2 bytes).

Values for the version 04 keyboard:

KEYCODE	FILL	US/UC	S only	C only	C/S	KEY NAME
$20	00	00	0A	14	1E	Function key 1
$21	00	01	0B	15	1F	Function key 2
$22	00	02	0C	16	20	Function key 3
$23	00	03	0D	17	21	Function key 4
$24	00	04	0E	18	22	Function key 5
$4A	00	FF	FF	FF	FF	COMMAND (closure)
$58	00	05	0F	19	23	Function key 6
$59	00	06	10	1A	24	Function key 7
$5A	00	07	11	1B	25	Function key 8
$5B	00	08	12	1C	26	Function key 9
$5C	00	09	13	1D	27	Function key 10
$CA	00	FE	FE	FE	FE	COMMMAND (release)

3.4 STANDARD MULTIPLE CHARACTER SEQUENCE TABLE (SMTABLE)

This table is used on key closure when a $9E table code is in the SHIFT TABLE (STABLE) or REGULAR TABLE (RLTABLE). Every entry with a $9E table code in the STABLE or RLTABLE must be in this table.

Each entry is composed of 3 fields. 1) the keycode, 2) the string length, and 3) the actual string. The string is the sequence of character codes placed in the buffer for this key. The Table does not have to be in keycode order. The table ends with a special keycode of $FF and length of 0.

Values for the version 04 keyboard:

KEYCODES	STRING LENGTH	STRING
$00 (cursor right)	2	$1B $43 (esc C)
$03 (HOME up)	2	$1B $48 (esc H)
$07 (enter)	2	$1B $64 (esc d)
$08 (cursor left)	2	$1B $44 (esc D)
$0B (cursor down)	2	$1B $42 (esc B)
$3A (back tab)	2	$1B $69 (esc i)
$5D (cursor up)	2	$1B $41 (esc A)
$4E (double zero)	2	$30 $30 (00)
$FF	0	END OF THE TABLE

3.5 CAPS LOCK & QUALIFIER FLAG TABLE (CQTABLE)

This table contains one byte for each keycode $00 - $5F. The Keycode is a direct index into the table. Each byte is a set of flags. All unused bits must be cleared (value = 0). The high order bit is the Caps lock flag for the corresponding Keycode. If the bit is set, this keycode generates a shifted character if the CAPS LOCK key is locked. Bit 6 is a special COMMAND key flag. The remaining bits are special key qualifier flags.

The bits currently defined are :

7 - Caps lock flag : when set means this keycode generates a shifted character when Caps lock is locked.

6 - Special COMMAND key flag:
- uses ETABLE for closure - keycode high order bit closure.
- uses ETABLE for release - keycode has high order bit set.
- special non-repeating key.

```
5 - Command -------
4 - Alternate      !        These bit indicate which type of
3 - Fast           !_______ special key the keycode represents.
2 - Caps lock      !        At most, one bit can be set on.
1 - Control        !
0 - Shift ---------
```

The values for the version 04 keyboard are listed in the attached program CSK.REV4.TEXT, listed in section 6.0.

3.6 RELEASE TABLE (RLTABLE)

This table specifies which keycodes have an action on key release. Each table has 2 fields. 1) the keycode, and 2) the action code.

The action code has 3 possible value types. If the action code is $9D it specifies a key with a ESCAPE # SEQUENCE TABLE (ETABLE) entry. If the action code is $9E it specifies a qualifier keycode. Any other action code is a character code to be placed into the buffer. The end of the table is specified by a special keycode of $FF and an action code of $00.

Values for the version 04 keyboard:

KEYCODE	ACTION CODE	KEY NAME
$1F	$9E	Right SHIFT
$3C	$9E	CAPS LOCK
$3E	$9E	Left SHIFT
$48	$9E	Control (CTRL)
$49	$9E	FAST
$4A	$9E	COMMAND
$4C	$9E	Alternate (ALT)
$FF	$00	NULL keycode - END OF TABLE

3.7 BREAK KEY CODE TABLE (BKEYCOD)

This table consists of one byte. It is the Keycode for the key which performs the start/stop toggle. The value for the version 04 keyboard is : $DF. This is the Keycode for BREAK closure.

4.0 Translation Table examples

This section gives the user several examples of how to change the Keyboard Tranlation tables. The examples deal with the unmarked key on the top row of keys (keycode $5E).

4.1 Alphabetic character example

The first example is to use the unmarked key (keycode $5E) as a standard alphabetic character key. This involves setting a value in the Translation Tables for the unshifted, shifted, and qualifier cases of the key.

A. These tables use the keycode value as an offset into the tables. Locate the unmarked key on the keyboard and note the position. Locate the same key in the keycode chart and note the keycode for closure (5E).

B. For this example let us assume the desired output of the Translations Tables is to be the alphabetic character 't' for unSHIFTed, 'T' for SHIFTed, and 'T' for CAPS LOCK.

C. Create a file with the same tables as the program CSK.REV4.TEXT.

D. Locate the position 5E in the SHIFT Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the SHIFT Table entry will be changed to a 'T' or 54 hex. Edit the STABLE at postion 5E hex to contain the value 54 hex.

THE SHIFT TABLE

THE SHIFT TABLE IS INDEXED BY KEYCODE. EACH BYTE REPRESENTS THE CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

The character symbol is above each character code

SMC	= special value for Standard Multiple Character Sequence	($9E)
QUL	= special value for Qualifier	($9F)
EST	= special value for Escape Sharp Character Sequence	($9D)
...	= No key for this keycode	

STABLE

```
          SMC  3    9    SMC  6    ,    -    cr   SMC  1    7    SMC  4    8    5    2    ;MSB
DATA.B    $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32  ;$00
          +    ...  {    del  cr   }    |    ...  )    ?    P    _    :    ~    "    QUL
DATA.B    $2B, $00, $7B, $7F, $0D, $7D, $7C, $00, $29, $3F, $50, $5F, $3A, $7E, $22, $9F  ;$10
          EST  EST  EST  EST  EST  ...  ...  ...  $    %    R    T    F    G    V    B
DATA.B    $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $24, $25, $52, $54, $46, $47, $56, $42  ;$20
          @    #    W    E    S    D    X    C    esc  !    SMC  Q    QUL  A    QUL  Z
DATA.B    $40, $23, $57, $45, $53, $44, $58, $43, $1B, $21, $9E, $51, $9F, $41, $9F, $5A  ;$30
          ^    &    Y    U    H    J    N    M    QUL  QUL  QUL  sp   QUL  O    SMC  .
DATA.B    $5E, $26, $59, $55, $48, $4A, $4E, $4D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E  ;$40
          *    (    I    O    K    L    <    >    EST  EST  EST  EST  EST  SMC  QUL  QUL
DATA.B    $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F  ;$50
LSB       0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

Change the last line to the following:

```
          *    (    I    O    K    L    <    >    EST  EST  EST  EST  EST  SMC  T    QUL
DATA.B    $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $54, $9F  ;$50
LSB       0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

E. Locate the position 5E in the REGULAR Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the REGULAR table entry will be changed to a 't' or 74 hex. Edit the RTABLE at position 5E hex to contain the value 74 hex.

THE REGULAR TABLE - UNSHIFTED OR LOWER CASE

THE REGULAR TABLE IS INDEXED BY KEYCODE. EACH BYTE REPRESENTS THE CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

The character symbol is above each character code

```
   SMC = special value for Standard Multiple Character Sequence    ($9E)
   QUL = special value for Qualifier                               ($9F)
   EST = special value for Escape Sharp Character Sequence         ($9D)
   ... = No key for this keycode
```

RTABLE

```
         SMC  3   9  SMC  6   ,   -   cr SMC  1   7  SMC 4   8   5   2  ;MSB
DATA.B  $9E,$33,$39,$9E,$36,$2C,$2D,$0D,$9E,$31,$37,$9E,$34,$38,$35,$32 ;$00
         =  ...   [  bs  cr   ]   \  ...  0   /   p   -   ;   `   '  QUL
DATA.B  $3D,$00,$5B,$08,$0D,$5D,$5C,$00,$30,$2F,$70,$2D,$3B,$60,$27,$9F ;$10
        EST EST EST EST EST ... ... ...   4   5   r   t   f   g   v   b
DATA.B  $9D,$9D,$9D,$9D,$9D,$00,$00,$00,$34,$35,$72,$74,$66,$67,$76,$62 ;$20
         2   3   w   e   s   d   x   c  esc  1  SMC  q  QUL  a  QUL  z
DATA.B  $32,$33,$77,$65,$73,$64,$78,$63,$1B,$31,$09,$71,$9F,$61,$9F,$7A ;$30
         6   7   y   u   h   j   n   m  QUL QUL QUL sp  QUL  0  SMC  .
DATA.B  $36,$37,$79,$75,$68,$6A,$6E,$6D,$9F,$9F,$9F,$20,$9F,$30,$9E,$2E ;$40
         8   9   i   o   k   l   ,   .  EST EST EST EST EST SMC QUL QUL
DATA.B  $38,$39,$69,$6F,$6B,$6C,$2C,$2E,$9D,$9D,$9D,$9D,$9D,$9E,$9F,$9F ;$50
LSB      0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
```

Change the last line to the following:

```
         8   9   i   o   k   l   ,   .  EST EST EST EST EST SMC  t  QUL
DATA.B  $38,$39,$69,$6F,$6B,$6C,$2C,$2E,$9D,$9D,$9D,$9D,$9D,$9E,$74,$9F ;$50
LSB      0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
```

F. Locate the position 5E in the CAPS/QUALIFIER Table. Note that the current entry is 00 hex which indicates the key does not have any flags set in the CAPS/QUALIFIER Table. In this example the CAPS/QUALIFIER Table entry will be changed to a 80 hex, to set the Caps lock flag in the table. Edit the CQTABLE at position 5E hex to contain the value 80 hex.

```
THE CAPS/QUALIFIER FLAG TABLE

 THE TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE ENTRY FOR
 THE CORRESPONDING KEYCODE.

 Each byte has 8 flags :
    D7 = Caps lock flag : when set means this keycode generates a
         shifted character when the Caps lock qualifier flag is set.
    D6 = Qualifier has an ESC # sequence flag.  When set then must process
         the keycode as a non-repeating ESC # sequence. Also has a Release
         sequence.

    D5 = Command -------
    D4 = Alternate     !
    D3 = Fast          !       This bit says which type of Qualifier
    D2 = Caps lock     !______ key the Keycode represents.
    D1 = Control       !
    D0 = Shift  _______!

CQTABLE
                                                                                        ; MSB
 DATA.B  $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00 ; $00
 DATA.B  $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $80, $00, $00, $00, $00, $01 ; $10
 DATA.B  $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $80, $80, $80, $80, $80, $80 ; $20
 DATA.B  $00, $00, $80, $80, $80, $80, $80, $80, $00, $00, $00, $80, $04, $80, $01, $80 ; $30
 DATA.B  $00, $00, $80, $80, $80, $80, $80, $80, $02, $08, $60, $00, $10, $00, $00, $00 ; $40
 DATA.B  $00, $00, $80, $80, $80, $80, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00 ; $50
  LSB     0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

Change the last line to the following:

```
 DATA.B  $00, $00, $80, $80, $80, $80, $00, $00, $00, $00, $00, $00, $00, $00, $80, $00 ; $50
 LSB      0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

G. Save the edited version of the Keyboard Translation Tables to a test file. Assemble the file as follows:

```
ASM68K filename [RETURN]
```

Upon completion of the assembly, link the file for quick load as follows:

```
LINKER filename [RETURN]
```

The last step is to load the new Keyboard Translation Table.

Press [WndowMgr].

Press [LdKybdCh].

Enter the filename, [RETURN].

A successful load of the tables will be noted in the Command Line. Begin testing the results of the new tables by pressing the unmarked key. Use the SHIFT key and the CAPS LOCK key and note the results.

4.2 STANDARD MULTIPLE CHARATER TABLE MODIFICATION

This example dealS with the modification of the STANDARD MULTIPLE CHARACTER SEQUENCE TABLE. The Translation Tables will now be modified to use the unmarked key (keycode 5E) as Cursor Right.

A. Create a file with the same entries as the CSK.REV4.TEXT file.

B. Locate the STANDARD MULTIPLE CHARACTER SEQUENCE TABLE within the file. It should be as follows:

```
 STANDARD MULTIPLE CHARACTER SEQUENCE TABLE
         FORMAT :   (KEYCODE, LENGTH, CHARACTER_SEQUENCE)

 The LENGTH field is the number of characters in the CHARACTER SEQUENCE field.
 The CHARACTER SEQUENCE is the characters to return for the Keycode.

SMTABLE
          KEYCODE LENGTH CHARACTER SEQUENCE
 DATA.B    $00,     2,     $1B,$43                 ;CURSOR RIGHT
 DATA.B    $03,     2,     $1B,$48                 ;HOME UP
 DATA.B    $07,     2,     $1B,$64                 ;ENTER
 DATA.B    $08,     2,     $1B,$44                 ;CURSOR LEFT
 DATA.B    $0B,     2,     $1B,$42                 ;CURSOR DOWN
 DATA.B    $3A,     2,     $1B,$69                 ;BACK TAB
 DATA.B    $5D,     2,     $1B,$41                 ;CURSOR UP
 DATA.B    $4E,     2,     $30,$30                 ;DOUBLE ZERO-( 00 KEY )
 DATA.B    $FF,     0                              ;NULL KEYCODE - END OF TABLE
```

C. Enter a duplication of the first entry in the table as the last entry in the table. Change the KEYCODE from $00 to $5E. The unmarked key is now defined as CURSOR RIGHT.

```
SMTABLE
          KEYCODE LENGTH CHARACTER SEQUENCE
 DATA.B    $00,     2,     $1B,$43                 ;CURSOR RIGHT
 DATA.B    $03,     2,     $1B,$48                 ;HOME UP
 DATA.B    $07,     2,     $1B,$64                 ;ENTER
 DATA.B    $08,     2,     $1B,$44                 ;CURSOR LEFT
 DATA.B    $0B,     2,     $1B,$42                 ;CURSOR DOWN
 DATA.B    $3A,     2,     $1B,$69                 ;BACK TAB
 DATA.B    $5D,     2,     $1B,$41                 ;CURSOR UP
 DATA.B    $4E,     2,     $30,$30                 ;DOUBLE ZERO-( 00 KEY )
 DATA.B    $5E,     2,     $1B,$43                 ;CURSOR RIGHT
 DATA.B    $FF,     0                              ;NULL KEYCODE - END OF TABLE
```

D. Locate the position 5E in the SHIFT Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the SHIFT Table entry will be changed to a $9E hex. Edit the STABLE at postion 5E hex to contain the value 9E hex.

THE SHIFT TABLE

THE SHIFT TABLE IS INDEXED BY KEYCODE. EACH BYTE REPRESENTS THE CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

```
The character symbol is above each character code
   SMC = special value for Standard Multiple Character Sequence   ($9E)
   QUL = special value for Qualifier                              ($9F)
   EST = special value for Escape Sharp Character Sequence        ($9D)
   ... = No key for this keycode
```

STABLE

```
         SMC  3    9    SMC  6    ,    -    cr   SMC  1    7    SMC  4    8    5    2    ;MSB
DATA.B   $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32  ;$00
         +    ...  {    del  cr   }    !    ...  )    ?    P    _    :    ~    "    QUL
DATA.B   $2B, $00, $7B, $7F, $0D, $7D, $7C, $00, $29, $3F, $50, $5F, $3A, $7E, $22, $9F  ;$10
         EST  EST  EST  EST  EST  ...  ...  ...  $    %    R    T    F    G    V    B
DATA.B   $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $24, $25, $52, $54, $46, $47, $56, $42  ;$20
         @    #    W    E    S    D    X    C    esc  !    SMC  Q    QUL  A    QUL  Z
DATA.B   $40, $23, $57, $45, $53, $44, $58, $43, $1B, $21, $9E, $51, $9F, $41, $9F, $5A  ;$30
         ^    &    Y    U    H    J    N    M    QUL  QUL  QUL  sp   QUL  O    SMC  .
DATA.B   $5E, $26, $59, $55, $48, $4A, $4E, $4D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E  ;$40
         *    (    I    O    K    L    <    >    EST  EST  EST  EST  EST  SMC  QUL  QUL
DATA.B   $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F  ,$50
LSB      0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

Change the last line to the following:

```
         *    (    I    O    K    L    <    >    EST  EST  EST  EST  EST  SMC  SMC  QUL
DATA.B   $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9E, $9F  ;$50
LSB      0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

E. Locate the position 5E in the REGULAR Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the REGULAR table entry will be changed to a 9E hex. Edit the RTABLE at position 5E hex to contain the value 9E hex.

THE REGULAR TABLE - UNSHIFTED OR LOWER CASE

THE REGULAR TABLE IS INDEXED BY KEYCODE. EACH BYTE REPRESENTS THE CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

```
 The character symbol is above each character code
    SMC = special value for Standard Multiple Character Sequence    ($9E)
    QUL = special value for Qualifier                               ($9F)
    EST = special value for Escape Sharp Character Sequence         ($9D)
    ... = No key for this keycode
```

RTABLE

```
         SMC  3    9    SMC  6    ,    -    cr   SMC  1    7    SMC  4    8    5    2   ;MSB
 DATA.B  $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32 ;$00
         =    ...  [    bs   cr   ]    \    ...  0    /    p    -    ;    `    '    QUL
 DATA.B  $3D, $00, $5B, $08, $0D, $5D, $5C, $00, $30, $2F, $70, $2D, $3B, $60, $27, $9F ;$10
         EST  EST  EST  EST  EST  ...  ...  ...  4    5    r    t    f    g    v    b
 DATA.B  $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $34, $35, $72, $74, $66, $67, $76, $62 ;$20
         2    3    w    e    s    d    x    c    esc  1    SMC  q    QUL  a    QUL  z
 DATA.B  $32, $33, $77, $65, $73, $64, $78, $63, $1B, $31, $09, $71, $9F, $61, $9F, $7A ;$30
         6    7    y    u    h    j    n    m    QUL  QUL  QUL  sp   QUL  0    SMC  .
 DATA.B  $36, $37, $79, $75, $68, $6A, $6E, $6D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E ;$40
         8    9    i    o    k    l    ,    .    EST  EST  EST  EST  EST  SMC  QUL  QUL
 DATA.B  $38, $39, $69, $6F, $6B, $6C, $2C, $2E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F ;$50
 LSB     0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

Change the last line to the following:

```
         8    9    i    o    k    l    ,    .    EST  EST  EST  EST  EST  SMC  SMC  QUL
 DATA.B  $38, $39, $69, $6F, $6B, $6C, $2C, $2E, $9D, $9D, $9D, $9D, $9D, $9E, $9E, $9F ;$50
 LSB     0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
```

F. Save, Assemble, Link, and Load as in the previous example.

4.3 ESCAPE SHARP SEQUENCE TABLE

This example deals with the modification of the ESCAPE SHARP SEQUENCE TABLE. The Translation Tables will now be modified to use the unmarked key as the FUNCTION KEY 1.

A. Create a file with the same entries as the CSK.REV4.TEXT file.

B. Locate the ESCAPE SHARP SEQUENCE TABLE within the file. It should be as follows:

```
 ESCAPE SHARP(#) SEQUENCE TABLE
        FORMAT :   (KEYCODE,FILLER,US/UC,SHIFT,COMMAND,C/S)
```

The fill field is added to keep each record on an even byte boundary The other fields contain the character sequence to follow the ESCAPE # characters:

```
   US/UC = when the Shift and Command key are released
   SHIFT = when only the Shift key is still being pressed
   COMMAND = when only the Command key is still being pressed
   C/S = when the Shift and Command keys are still being pressed
```

```
ETABLE
         KEYCODE FILL    US/UC   SHIFT   COMMAND    C/S
  DATA.B  $20,     0,    '00',   '0A',   '14',      '1E' ;FUNCTION KEY 1
  DATA.B  $21,     0,    '01',   '0B',   '15',      '1F' ;FUNCTION KEY 2
  DATA.B  $22,     0,    '02',   '0C',   '16',      '20' ;FUNCTION KEY 3
  DATA.B  $23,     0,    '03',   '0D',   '17',      '21' ;FUNCTION KEY 4
  DATA.B  $24,     0,    '04',   '0E',   '18',      '22' ;FUNCTION KEY 5
  DATA.B  $4A,     0,    'FF',   'FF',   'FF',      'FF' ;LEFT COMMAND (CLOSURE)
  DATA.B  $58,     0,    '05',   '0F',   '19',      '23' ;FUNCTION KEY 6
  DATA.B  $59,     0,    '06',   '10',   '1A',      '24' ;FUNCTION KEY 7
  DATA.B  $5A,     0,    '07',   '11',   '1B',      '25' ;FUNCTION KEY 8
  DATA.B  $5B,     0,    '08',   '12',   '1C',      '26' ;FUNCTION KEY 9
  DATA.B  $5C,     0,    '09',   '13',   '1D',      '27' ;FUNCTION KEY 10
  DATA.B  $CA,     0,    'FE',   'FE',   'FE',      'FE' ;LEFT COMMAND (RELEASE)
```

C. Enter a duplication of the first entry in the table as the last entry in the table. Change the KEYCODE from $20 to $5E. The unmarked key is no defined as FUNCTION KEY 1.

```
ETABLE
         KEYCODE FILL    US/UC   SHIFT   COMMAND    C/S
  DATA.B  $20,     0,    '00',   '0A',   '14',      '1E' ;FUNCTION KEY 1
  DATA.B  $21,     0,    '01',   '0B',   '15',      '1F' ;FUNCTION KEY 2
  DATA.B  $22,     0,    '02',   '0C',   '16',      '20' ;FUNCTION KEY 3
  DATA.B  $23,     0,    '03',   '0D',   '17',      '21' ;FUNCTION KEY 4
  DATA.B  $24,     0,    '04',   '0E',   '18',      '22' ;FUNCTION KEY 5
DATA.B  $4A,    0,    'FF',   'FF',   'FF',    'FF' ;LEFT COMMAND (CLOSURE)
DATA.B  $58,    0,    '05',   '0F',   '19',    '23' ;FUNCTION KEY 6
DATA.B  $59,    0,    '06',   '10',   '1A',    '24' ;FUNCTION KEY 7
DATA.B  $5A,    0,    '07',   '11',   '1B',    '25' ;FUNCTION KEY 8
DATA.B  $5B,    0,    '08',   '12',   '1C',    '26' ;FUNCTION KEY 9
DATA.B  $5C,    0,    '09',   '13',   '1D',    '27' ;FUNCTION KEY 10
DATA.B  $5E,    0,    '00',   '0A',   '14',    '1E' ;FUNCTION KEY 1
DATA.B  $CA,    0,    'FE',   'FE',   'FE',    'FE' ;LEFT COMMAND (RELEASE)
```

D. Locate the position 5E in the SHIFT Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the SHIFT Table entry will be changed to a $9D hex. Edit the STABLE at postion 5E hex to contain the value 9D.

THE SHIFT TABLE

```
 THE SHIFT TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE
 CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

 The character symbol is above each character code
    SMC = special value for Standard Multiple Character Sequence   ($9E)
    QUL = special value for Qualifier                              ($9F)
    EST = special value for Escape Sharp Character Sequence        ($9D)
    ... = No key for this keycode

STABLE

          SMC  3   9  SMC  6   ,   -  cr SMC  1   7  SMC 4   8   5   2  ;MSB
 DATA.B   $9E,$33,$39,$9E,$36,$2C,$2D,$0D,$9E,$31,$37,$9E,$34,$38,$35,$32 ;$00
           +  ...  {  del  cr  }   |  ...  )   ?   P   _   :   ~   "  QUL
 DATA.B   $2B,$00,$7B,$7F,$0D,$7D,$7C,$00,$29,$3F,$50,$5F,$3A,$7E,$22,$9F ;$10
          EST EST EST EST EST ... ... ...  $   %   R   T   F   G   V   B
 DATA.B   $9D,$9D,$9D,$9D,$9D,$00,$00,$00,$24,$25,$52,$54,$46,$47,$56,$42 ;$20
           @   #   W   E   S   D   X   C  esc  !  SMC  Q  QUL  A  QUL  Z
 DATA.B   $40,$23,$57,$45,$53,$44,$58,$43,$1B,$21,$9E,$51,$9F,$41,$9F,$5A ;$30
           ^   &   Y   U   H   J   N   M  QUL QUL QUL sp  QUL  O  SMC  .
 DATA.B   $5E,$26,$59,$55,$48,$4A,$4E,$4D,$9F,$9F,$9F,$20,$9F,$30,$9E,$2E ;$40
           *   (   I   O   K   L   <   >  EST EST EST EST EST SMC QUL QUL
 DATA.B   $2A,$28,$49,$4F,$4B,$4C,$3C,$3E,$9D,$9D,$9D,$9D,$9D,$9E,$9F,$9F ;$50
 LSB       0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F


             Change the last line to the following:

           *   (   I   O   K   L   <   >  EST EST EST EST EST SMC EST QUL
 DATA.B   $2A,$28,$49,$4F,$4B,$4C,$3C,$3E,$9D,$9D,$9D,$9D,$9D,$9E,$9D,$9F   ;$50
 LSB       0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
```

E. Locate the position 5E in the REGULAR Table. Note that the current entry is 9F hex which indicates the key is a qualifier. In this example the REGULAR table entry will be changed to a $9D hex. Edit the RTABLE at position 5E hex to contain the value 9D hex.

```
THE REGULAR TABLE - UNSHIFTED OR LOWER CASE

 THE REGULAR TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE
 CHARACTER CODE FOR THE CORRESPONDING KEYCODE.

 The character symbol is above each character code
    SMC = special value for Standard Multiple Character Sequence    ($9E)
    QUL = special value for Qualifier                               ($9F)
    EST = special value for Escape Sharp Character Sequence         ($9D)
    ... = No key for this keycode

RTABLE

          SMC  3   9   SMC  6   ,   -   cr  SMC  1   7   SMC  4   8   5   2  ;MSB
 DATA.B   $9E,$33,$39,$9E,$36,$2C,$2D,$0D,$9E,$31,$37,$9E,$34,$38,$35,$32 ;$00
           =  ...  [   bs  cr  ]   \  ...  0   /   p   -   ;   `   '  QUL
 DATA.B   $3D,$00,$5B,$08,$0D,$5D,$5C,$00,$30,$2F,$70,$2D,$3B,$60,$27,$9F ;$10
          EST EST EST EST EST ... ... ...  4   5   r   t   f   g   v   b
 DATA.B   $9D,$9D,$9D,$9D,$9D,$00,$00,$00,$34,$35,$72,$74,$66,$67,$76,$62 ;$20
           2   3   w   e   s   d   x   c  esc  1  SMC  q  QUL  a  QUL  z
 DATA.B   $32,$33,$77,$65,$73,$64,$78,$63,$1B,$31,$09,$71,$9F,$61,$9F,$7A ;$30
           6   7   y   u   h   j   n   m  QUL QUL QUL  sp QUL  0  SMC  .
 DATA.B   $36,$37,$79,$75,$68,$6A,$6E,$6D,$9F,$9F,$9F,$20,$9F,$30,$9E,$2E ;$40
           8   9   i   o   k   l   ,   .  EST EST EST EST EST SMC QUL QUL
 DATA.B   $38,$39,$69,$6F,$6B,$6C,$2C,$2E,$9D,$9D,$9D,$9D,$9D,$9E,$9F,$9F ;$50
 LSB       0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F


            Change the last line to the following:

           8   9   i   o   k   l   ,   .  EST EST EST EST EST SMC EST QUL
 DATA.B   $38,$39,$69,$6F,$6B,$6C,$2C,$2E,$9D,$9D,$9D,$9D,$9D,$9E,$9D,$9F ;$50
 LSB       0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
```

F. Save, Assemble, Link, and Load as in the previous example.

5.0 Default Keyboard Translation Table

To make a Keyboard Translation Table the system default table, put the linker out file for the Keyboard Translation Table into the volume /CCSYS/ with a file name of CSK.DEFAULT.

6.0 Program CSK.REV4.TEXT listing

```
; THIS FILE CONTAINS THE TABLES FOR THE KEYBOARD DRIVER FOR THE VERSION
; 04 KEYBOARD (Selectric style ASCII with ALT key and Back Space key
; moved from the version 3 location).
;
; file : csk.rev4.text
; date : 05-Oct-82  kb
;
; TRANSLATION TABLE
;
TRANTBL
            DATA.L     STABLE - TRANTBL  ;POINTER TO SHIFT TABLE
            DATA.L     RTABLE - TRANTBL  ;POINTER TO REGULAR TABLE
            DATA.L     ETABLE - TRANTBL  ;POINTER TO ESCAPE # TABLE
            DATA.L     SMTABLE- TRANTBL  ;PTR TO STANDARD MULT CHAR TABLE
            DATA.L     CQTABLE- TRANTBL  ;POINTER TO CAP/QUALIFIER TABLE
            DATA.L     RLTABLE- TRANTBL  ;POINTER TO RELEASE TABLE
            DATA.L     BKEYCOD- TRANTBL  ;POINTER TO BREAK KEYCODE TABLE
;
; LENGTH OF FILE DATA AFTER TRANSLATION TABLE
;
            DATA.W     LENGTH
;
; VERSION DATE
;
VERSION     DATA.B     '070682'
;
START       RTS
  page
;
; NOTE:
; All the tables have keycodes with the closure/release bit (MSB) of the Keycod
; clear (0), except the Break Keycode Table.
;
; THE SHIFT TABLE
;        TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;        CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is being pressed (Shift flag is set).
;
;  The character symbol is above each character code
;    SMC = special value for Standard Multiple Character Sequence ($9E)
;    QUL = special value for Qualifier                            ($9F)
;    EST = special value for Escape Sharp Character Sequence      ($9D)
;    ... = No key for this keycode
;
```

```
STABLE

;           SMC   3    9   SMC   6    ,    -    cr  SMC   1    7   SMC   4    8    5    2  ;MSB
  DATA. B   $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32 ;$00
;            +   ...   {   del   cr   }    |   ...   )    ?    P    _    :    ~    "   QUL
  DATA. B   $2B, $00, $7B, $7F, $0D, $7D, $7C, $00, $29, $3F, $50, $5F, $3A, $7E, $22, $9F ;$10
;           EST  EST  EST  EST  EST  ...  ...  ...   $    %    R    T    F    G    V    B
  DATA. B   $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $24, $25, $52, $54, $46, $47, $56, $42 ;$20
;            @    #    W    E    S    D    X    C   esc   !   SMC   Q   QUL   A   QUL   Z
  DATA. B   $40, $23, $57, $45, $53, $44, $58, $43, $1B, $21, $9E, $51, $9F, $41, $9F, $5A ;$30
;            ^    &    Y    U    H    J    N    M   QUL  QUL  QUL   sp  QUL   0   SMC   .
  DATA. B   $5E, $26, $59, $55, $48, $4A, $4E, $4D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E ;$40
;            *    (    I    O    K    L ,  <    >   EST  EST  EST  EST  EST  SMC  EST  EST
  DATA. B   $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F ;$50
; LSB        0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
  page
;
; THE REGULAR TABLE -- UNSHIFTED OR LOWER CASE
;        TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;        CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is released (Shift flag is clear).
;
;  The character symbol is above each character code
;    SMC = special value for Standard Multiple Character Sequence ($9E)
;    QUL = special value for Qualifier                            ($9F)
;    EST = special value for Escape Sharp Character Sequence      ($9D)
;    ... = No key for this keycode
;
RTABLE

;           SMC   3    9   SMC   6    ,    -    cr  SMC   1    7   SMC   4    8    5    2  ;MSB
  DATA. B   $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32 ;$00
;            =   ...   [    bs   cr   ]    \   ...   0    /    p    -    ;    `    '   QUL
  DATA. B   $3D, $00, $5B, $08, $0D, $5D, $5C, $00, $30, $2F, $70, $2D, $3B, $60, $27, $9F ;$10
;           EST  EST  EST  EST  EST  ...  ...  ...   4    5    r    t    f    g    v    b
  DATA. B   $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $34, $35, $72, $74, $66, $67, $76, $62 ;$20
;            2    3    w    e    s    d    x    c   esc   1   SMC   q   QUL   a   QUL   z
  DATA. B   $32, $33, $77, $65, $73, $64, $78, $63, $1B, $31, $09, $71, $9F, $61, $9F, $7A ;$30
;            6    7    y    u    h    j    n    m   QUL  QUL  QUL   sp  QUL   0   SMC   .
  DATA. B   $36, $37, $79, $75, $68, $6A, $6E, $6D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E ;$40
;            8    9    i    o    k    l    ,    .   EST  EST  EST  EST  EST  SMC  EST  EST
  DATA. B   $38, $39, $69, $6F, $6B, $6C, $2C, $2E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F ;$50
; LSB        0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
  page
;
; THE CAPS/QUALIFIER FLAG TABLE
;        TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE ENTRY FOR
;        THE CORRESPONDING KEYCODE.
;
;  Used on Closure when a $9F is in the Keycode entry of the Shift or Regular
;       Table, process a Qualifier.  Also on Closure when the Caps Lock Key is
;        being pressed (CapsLock flag is set).
;   Used on Release when a $9E action code is in the Keycode entry of the Releas
;        Table.
;
;   Each byte has 8 flags :
;        D7 = Caps lock flag : when set means this keycode generates a
;             shifted character when the Caps lock qualifier flag is set.
;        D6 = Qualifier has an ESC # sequence flag.  When set then must process
;             the keycode as a non-repeating ESC # sequence. Also has a Release
;             sequence.
;
;        D5 = Command -------
;        D4 = Alternate      |
;        D3 = Fast           |       This bit says which type of Qualifier
;        D2 = Caps lock      |______ key the Keycode represents.
;        D1 = Control        |
;        D0 = Shift   _______|
;
CQTABLE
                                                                                      ;MSB
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$00
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$00,$00,$00,$00,$01 ;$10
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$80,$80,$80,$80,$80 ;$20
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$00,$00,$00,$80,$04,$80,$01,$80 ;$30
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$02,$08,$60,$00,$10,$00,$00,$00 ;$40
  DATA.B  $00,$00,$80,$80,$80,$80,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$50
; LSB      0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
  page
; ESCAPE SHARP(#) SEQUENCE TABLE
;         FORMAT :   (KEYCODE,FILLER,US/UC,SHIFT,COMMAND,C/S)
;
; Used on Closure when a $9D is in the Keycode entry of the Shift or
;       Regular Table.
; Used on Release when a $9D action code is in the Keycode entry of
;       the Release Table.  Release keycode has high order bit set.
;
; The fill field is added to keep each record on an even byte boundary
; The other fields contain the character sequence to follow the ESCAPE #
;       characters:
;       US/UC = when the Shift and Command key are released
;       SHIFT = when only the Shift key is still being pressed
;     COMMAND = when only the Command key is still being pressed
;         C/S = when the Shift and Command keys are still being pressed
;
ETABLE
;         KEYCODE FILL    US/UC   SHIFT   COMMAND    C/S
  DATA.B  $20,      0,    '00',   '0A',   '14',      '1E' ;FUNCTION KEY 1
  DATA.B  $21,      0,    '01',   '0B',   '15',      '1F' ;FUNCTION KEY 2
  DATA.B  $22,      0,    '02',   '0C',   '16',      '20' ;FUNCTION KEY 3
  DATA.B  $23,      0,    '03',   '0D',   '17',      '21' ;FUNCTION KEY 4
  DATA.B  $24,      0,    '04',   '0E',   '18',      '22' ;FUNCTION KEY 5
  DATA.B  $4A,    0,     'FF',   'FF',   'FF',      'FF' ;LEFT COMMAND (CLOSURE)
  DATA.B  $58,    0,     '05',   '0F',   '19',      '23' ;FUNCTION KEY 6
  DATA.B  $59,    0,     '06',   '10',   '1A',      '24' ;FUNCTION KEY 7.
  DATA.B  $5A,    0,     '07',   '11',   '1B',      '25' ;FUNCTION KEY 8
  DATA.B  $5B,    0,     '08',   '12',   '1C',      '26' ;FUNCTION KEY 9
  DATA.B  $5C,    0,     '09',   '13',   '1D',      '27' ;FUNCTION KEY 10
  DATA.B  $CA,    0,     'FE',   'FE',   'FE',      'FE' ;LEFT COMMAND (RELEASE)
  page
;
; STANDARD MULTIPLE CHARACTER SEQUENCE TABLE
;        FORMAT :   (KEYCODE, LENGTH, CHARACTER_SEQUENCE)
;
; Used on Closure when a $9E is in the Keycode entry of the Shift or
;       Regular Table.
;
; The LENGTH field is the number of characters in the CHARACTER SEQUENCE field.
; The CHARACTER SEQUENCE is the characters to return for the Keycode.
;
SMTABLE
;         KEYCODE LENGTH CHARACTER SEQUENCE
  DATA.B    $00,     2,     $1B,$43    ;CURSOR RIGHT
  DATA.B    $03,     2,     $1B,$48    ;HOME UP
  DATA.B    $07,     2,     $1B,$64    ;ENTER
  DATA.B    $08,     2,     $1B,$44    ;CURSOR LEFT
  DATA.B    $0B,     2,     $1B,$42    ;CURSOR DOWN
  DATA.B    $3A,     2,     $1B,$69    ;BACK TAB
  DATA.B    $5D,     2,     $1B,$41    ;CURSOR UP
  DATA.B    $4E,     2,     $30,$30    ;DOUBLE ZERO-( 00 KEY )
  DATA.B    $FF,     0                 ;NULL KEYCODE - END OF TABLE
  page
;
; RELEASE TABLE
;        FORMAT :   (KEYCODE, ACTION_CODE)
;
; Used on all Release keycodes
;
; The action code describes the type of key:
;       9D = return an Escape Sharp Sequence for this keycode
;       9E = a Qualifier key
;all other = character code to return
;
RLTABLE
;         KEYCODE          ACTION CODE
  DATA.B    $1F,               $9E    ;RIGHT SHIFT
  DATA.B    $3C,               $9E    ;CAPS LOCK
  DATA.B    $3E,               $9E    ;LEFT SHIFT
  DATA.B    $48,               $9E    ;CONTROL
  DATA.B    $49,               $9E    ;FAST
  DATA.B    $4A,               $9E    ;LEFT COMMAND
  DATA.B    $4C,               $9E    ;ALTERNATE
  DATA.B    $FF,               $00    ;NULL KEYCODE - END OF TABLE
```

```
;
;
; BREAK KEYCODE TABLE
;         SINGLE BYTE TABLE.   THIS IS THE KEYCODE WHICH CAUSES START/STOP.
;
; Used on all keycodes.
;
; The filler is to keep the file on an even byte boundary
;
BKEYCOD  DATA.B     $DF,0         ;BREAK CLOSURE KEYCODE,FILLER
;
LENGTH   EQU        %-VERSION     ;LENGTH OF DATA AFTER TRANSLATION TABLE
         END        START
```

7.0 Program CSK.DANSK listing

```
;
; THIS FILE CONTAINS THE TABLES FOR THE KEYBOARD DRIVER FOR THE VERSION
; 04 Danish KEYBOARD (Selectric style with ALT key).
;
; NOTE:
; Because this document was printed on a standard ASCII printer,
; special Danish characters are printed as ASCII characters.
;
; file : csk.dansk.text
; date : 05-Oct-82  kb
;
; TRANSLATION TABLE
;
TRANTBL
            DATA.L      STABLE - TRANTBL    ;POINTER TO SHIFT TABLE
            DATA.L      RTABLE - TRANTBL    ;POINTER TO REGULAR TABLE
            DATA.L      ETABLE - TRANTBL    ;POINTER TO ESCAPE # TABLE
            DATA.L      SMTABLE- TRANTBL    ;PTR TO STANDARD MULT CHAR TABLE
            DATA.L      CQTABLE- TRANTBL    ;POINTER TO CAP/QUALIFIER TABLE
            DATA.L      RLTABLE- TRANTBL    ;POINTER TO RELEASE TABLE
            DATA.L      BKEYCOD- TRANTBL    ;POINTER TO BREAK KEYCODE TABLE
;
; LENGTH OF FILE DATA AFTER TRANSLATION TABLE
;
            DATA.W      LENGTH
;
; VERSION DATE
;
VERSION     DATA.B      '051082'    ;ddmmyy - day month year
;
START       RTS
  page
;
; NOTE:
;
; All the tables have keycodes with the closure/release bit (MSB) of the
; Keycode clear (0), except the Break Keycode Table.
;
; THE SHIFT TABLE
;   TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;   CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is still depressed (Shift flag is set)
;
;  The character symbol is above each character code
;    SMC = special value for Standard Multiple Character Sequence ($9E)
;    QUL = special value for Qualifier                            ($9F)
;    EST = special value for Escape Sharp Character Sequence      ($9D)
;     ... = No key for this keycode
;
STABLE

;           SMC  3   9  SMC  6   ,   -   cr SMC  1   7   SMC 4   8   5   2  ;MSB
  DATA.B    $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32 ; $00
;            +  ...  "  del  cr  :   ~  ...  )   ?   P   _   L   S   R  QUL
  DATA.B    $2B, $00, $22, $7F, $0D, $3A, $7E, $00, $29, $3F, $50, $5F, $5B, $5D, $5C, $9F ; $10
;           EST EST EST EST EST ... ... ...  $   %   R   T   F   G   V   B
  DATA.B    $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $24, $25, $52, $54, $46, $47, $56, $42 ; $20
;            @   #   W   E   S   D   X   C  esc  !  SMC  Q  QUL  A  QUL  Y
  DATA.B    $40, $23, $57, $45, $53, $44, $58, $43, $1B, $21, $9E, $51, $9F, $41, $9F, $59 ; $30
;            ^   &   Z   U   H   J   N   M  QUL QUL QUL sp  QUL  O  SMC  .
  DATA.B    $5E, $26, $5A, $55, $48, $4A, $4E, $4D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E ; $40
;            *   (   I   O   K   L   <   >  EST EST EST EST EST SMC EST EST
  DATA.B    $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F ; $50
; LSB        0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
  page
;
; THE REGULAR TABLE - UNSHIFTED OR LOWER CASE
;        TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;        CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is released (Shift flag is clear).
;
;  The character symbol is above each character code
;       SMC = special value for Standard Multiple Character Sequence ($9E)
;       QUL = special value for Qualifier                            ($9F)
;       EST = special value for Escape Sharp Character Sequence      ($9D)
;       ... = No key for this keycode
;
RTABLE

;           SMC  3   9  SMC  6   ,   -   cr SMC  1   7   SMC 4   8   5   2  ;MSB
  DATA.B    $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32 ; $00
;            =  ...  '   bs  cr  ;   `  ...  0   /   p   -   l   s   r  QUL
  DATA.B    $3D, $00, $27, $08, $0D, $3B, $60, $00, $30, $2F, $70, $2D, $7B, $7D, $7C, $9F ; $10
;           EST EST EST EST EST ... ... ...  4   5   r   t   f   g   v   b
  DATA.B    $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $34, $35, $72, $74, $66, $67, $76, $62 ; $20
;            2   3   w   e   s   d   x   c  esc  1  SMC  q  QUL  a  QUL  y
  DATA.B    $32, $33, $77, $65, $73, $64, $78, $63, $1B, $31, $09, $71, $9F, $61, $9F, $79 ; $30
;            6   7   z   u   h   j   n   m  QUL QUL QUL sp  QUL  0  SMC  .
  DATA.B    $36, $37, $7A, $75, $68, $6A, $6E, $6D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E ; $40
;            8   9   i   o   k   l   ,   .  EST EST EST EST EST SMC EST EST
  DATA.B    $38, $39, $69, $6F, $6B, $6C, $2C, $2E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F ; $50
; LSB        0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
  page
;
; THE CAPS/QUALIFIER FLAG TABLE
;        TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE ENTRY FOR
;        THE CORRESPONDING KEYCODE.
```

```
;
;  Used on Closure when a $9F is in the Keycode entry of the Shift or Regular
;       Table, process a Qualifier.  Also on Closure when the Caps Lock Key is
;       being pressed (CapsLock flag is set).
,  Used on Release when a $9E action code is in the Keycode entry of the Releas
;       Table.
,
;  Each byte has 8 flags :
;       D7 = Caps lock flag : when set means this keycode generates a
,            shifted character when the Caps lock qualifier flag is set.
;       D6 = Qualifier has an ESC # sequence flag.  When set then must process
;            the keycode as a non-repeating ESC # sequence. Also has a Release
;          . sequence.
,
;       D5 = Command -------
,       D4 = Alternate      !
,       D3 = Fast           !       This bit says which type of Qualifier
;       D2 = Caps lock      !_____  key the Keycode represents.
;       D1 = Control        !
;       DO = Shift  ________!
;
CQTABLE
                                                                                    ;MSB
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$00
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$00,$00,$00,$00,$01 ;$10
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$80,$80,$80,$80,$80 ;$20
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$00,$00,$00,$80,$04,$80,$01,$80 ;$30
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$02,$08,$60,$00,$10,$00,$00,$00 ;$40
  DATA.B  $00,$00,$80,$80,$80,$80,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$50
, LSB      0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
  page
; ESCAPE SHARP(#) SEQUENCE TABLE
;        FORMAT :   (KEYCODE,FILLER,US/UC,SHIFT,COMMAND,C/S)
;
, Used on Closure when a $9D is in the Keycode entry of the Shift or
;       Regular Table.
; Used on Release when a $9D action code is in the Keycode entry of
;       the Release Table.  Release keycode has high order bit set.
;
; The fill field is added to keep each record on an even byte boundary
; The other fields contain the character sequence to follow the ESCAPE #
;       characters:
,       US/UC = when the Shift and Command key are not pressed
;       SHIFT = when only the Shift key is still being pressed
;     COMMAND = when only the Command key is still being pressed
;         C/S = when the Shift and Command keys are still being pressed
,
ETABLE
;        KEYCODE FILL   US/UC  SHIFT  COMMAND   C/S
  DATA.B  $20,     0,    '00',  '0A',  '14',     '1E' ;FUNCTION KEY 1
  DATA.B  $21,     0,    '01',  '0B',  '15',     '1F' ;FUNCTION KEY 2
  DATA.B  $22,     0,     '02',   '0C',   '16',     '20' ;FUNCTION KEY 3
  DATA.B  $23,     0,     '03',   '0D',   '17',     '21' ;FUNCTION KEY 4
  DATA.B  $24,     0,     '04',   '0E',   '18',     '22' ;FUNCTION KEY 5
  DATA.B  $4A,     0,     'FF',   'FF',   'FF',     'FF' ;LEFT COMMAND (CLOSURE)
  DATA.B  $58,     0,     '05',   '0F',   '19',     '23' ;FUNCTION KEY 6
  DATA.B  $59,     0,     '06',   '10',   '1A',     '24' ;FUNCTION KEY 7
  DATA.B  $5A,     0,     '07',   '11',   '1B',     '25' ;FUNCTION KEY 8
  DATA.B  $5B,     0,     '08',   '12',   '1C',     '26' ;FUNCTION KEY 9
  DATA.B  $5C,     0,     '09',   '13',   '1D',     '27' ;FUNCTION KEY 10
  DATA.B  $CA,     0,     'FE',   'FE',   'FE',     'FE' ;LEFT COMMAND (RELEASE)
  page
;
; STANDARD MULTIPLE CHARACTER SEQUENCE TABLE
;         FORMAT :   (KEYCODE, LENGTH, CHARACTER_SEQUENCE)
;
; Used on Closure when a $9E is in the Keycode entry of the Shift or
;       Regular Table.
;
; The LENGTH field is the number of characters in the CHARACTER SEQUENCE field
; The CHARACTER SEQUENCE is the characters to return for the Keycode.
;
SMTABLE
;          KEYCODE LENGTH CHARACTER SEQUENCE
  DATA.B     $00,     2,      $1B,$43    ;CURSOR RIGHT
  DATA.B     $03,     2,      $1B,$48    ;HOME UP
  DATA.B     $07,     2,      $1B,$64    ;ENTER
  DATA.B     $08,     2,      $1B,$44    ;CURSOR LEFT
  DATA.B     $0B,     2,      $1B,$42    ;CURSOR DOWN
  DATA.B     $3A,     2,      $1B,$69    ;BACK TAB
  DATA.B     $5D,     2,      $1B,$41    ;CURSOR UP
  DATA.B     $4E,     2,      $30,$30    ;DOUBLE ZERO-( 00 KEY )
  DATA.B     $FF,     0                  ;NULL KEYCODE - END OF TABLE
  page
;
; RELEASE TABLE
;         FORMAT :   (KEYCODE, ACTION_CODE)
;
; Used on all Release keycodes.
;
; The action code describes the type of key:
;       9D = return an Escape Sharp Sequence for this keycode
;       9E = a Qualifier key
;all other = character code to return
;
RLTABLE
;          KEYCODE        ACTION CODE
  DATA.B     $1F,              $9E    ;RIGHT SHIFT
  DATA.B     $3C,              $9E    ;CAPS LOCK
  DATA.B     $3E,              $9E    ;LEFT SHIFT
  DATA.B     $48,              $9E    ;CONTROL
  DATA.B     $49,              $9E    ;FAST
```

```
   DATA.B      $4A,                  $9E    ;LEFT COMMAND
   DATA.B      $4C,                  $9E    ;ALTERNATE
   DATA.B      $FF,                  $00    ;NULL KEYCODE - END OF TABLE
;
;
; BREAK KEYCODE TABLE
;          SINGLE BYTE TABLE.   THIS IS THE KEYCODE WHICH CAUSES START/STOP.
;
; Used on all keycodes.
;
; The filler is to keep the file on an even byte boundary
;
BKEYCOD  DATA.B     $DF,0          ;BREAK CLOSURE KEYCODE,FILLER
;
LENGTH   EQU        %-VERSION      ;LENGTH OF DATA AFTER TRANSLATION TABLE
         END        START
```

8.0 Program CSK.GRMN.TEXT listing

```
;
; THIS FILE CONTAINS THE TABLES FOR THE KEYBOARD DRIVER FOR THE VERSION
; 03 German KEYBOARD (Selectric style with ALT key).
;
; NOTE:
; Because this document was printed on a standard ASCII printer,
; special German characters are printed as ASCII characters.
;
; file : csk.grmn.text
; date : 05-Oct-82  kb
;
; TRANSLATION TABLE
;
TRANTBL
              DATA.L     STABLE - TRANTBL   ;POINTER TO SHIFT TABLE
              DATA.L     RTABLE - TRANTBL   ;POINTER TO REGULAR TABLE
              DATA.L     ETABLE - TRANTBL   ;POINTER TO ESCAPE # TABLE
              DATA.L     SMTABLE- TRANTBL   ;PTR TO STANDARD MULT CHAR TABLE
              DATA.L     CQTABLE- TRANTBL   ;POINTER TO CAP/QUALIFIER TABLE
              DATA.L     RLTABLE- TRANTBL   ;POINTER TO RELEASE TABLE
              DATA.L     BKEYCOD- TRANTBL   ;POINTER TO BREAK KEYCODE TABLE
;
; LENGTH OF FILE DATA AFTER TRANSLATION TABLE
;
              DATA.W     LENGTH
;
; VERSION DATE
;
VERSION       DATA.B     '051082'   ;ddmmyy - day month year
;
START         RTS
  page
;
; NOTE:
;
; All the tables have keycodes with the closure/release bit (MSB) of the
; Keycode clear (0), except the Break Keycode Table.
;
; THE SHIFT TABLE
;         TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;         CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is still depressed (Shift flag is set).
;
;  The character symbol is above each character code
;    SMC = special value for Standard Multiple Character Sequence ($9E)
;    QUL = special value for Qualifier                            ($9F)
;    EST = special value for Escape Sharp Character Sequence      ($9D)
;     ... = No key for this keycode
;
STABLE

;          SMC  3    9    SMC  6    ,    -    cr   SMC  1    7    SMC  4    8    5    2    ;MSB
  DATA.B   $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32  ;$00
;          +    ...  "    del  cr   :    `    ...  )    ?    P    _    O    U    A    QUL
  DATA.B   $2B, $00, $22, $7F, $0D, $3A, $60, $00, $29, $3F, $50, $5F, $5C, $5D, $5B, $9F  ;$10
;          EST  EST  EST  EST  EST  ...  ...  ...  $    %    R    T    F    G    V    B
  DATA.B   $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $24, $25, $52, $54, $46, $47, $56, $42  ;$20
;          [    #    W    E    S    D    X    C    esc  !    SMC  Q    QUL  A    QUL  Y
  DATA.B   $40, $23, $57, $45, $53, $44, $58, $43, $1B, $21, $9E, $51, $9F, $41, $9F, $59  ;$30
;          ^    &    Z    U    H    J    N    M    QUL  QUL  QUL  sp   QUL  O    SMC  .
  DATA.B   $5E, $26, $5A, $55, $48, $4A, $4E, $4D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E  ;$40
;          *    (    I    O    K    L    <    >    EST  EST  EST  EST  EST  SMC  EST  EST
  DATA.B   $2A, $28, $49, $4F, $4B, $4C, $3C, $3E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F  ;$50
; LSB      0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
  page
;
; THE REGULAR TABLE - UNSHIFTED OR LOWER CASE
;         TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE CHARACTER
;         CODE FOR THE CORRESPONDING KEYCODE.
;
;  Used on Closure only when Shift key is released (Shift flag is clear).
;
;  The character symbol is above each character code
;    SMC = special value for Standard Multiple Character Sequence  ($9E)
;    QUL = special value for Qualifier                             ($9F)
;    EST = special value for Escape Sharp Character Sequence       ($9D)
;    ... = No key for this keycode
;
RTABLE

;          SMC  3    9    SMC  6    ,    -    cr   SMC  1    7    SMC  4    8    5    2    ;MSB
  DATA.B   $9E, $33, $39, $9E, $36, $2C, $2D, $0D, $9E, $31, $37, $9E, $34, $38, $35, $32  ;$00
;          =    ...  '    bs   cr   ;    z    ...  0    /    p    -    o    u    a    QUL
  DATA.B   $3D, $00, $27, $08, $0D, $3B, $7E, $00, $30, $2F, $70, $2D, $7C, $7D, $7B, $9F  ;$10
;          EST  EST  EST  EST  EST  ...  ...  ...  4    5    r    t    f    g    v    b
  DATA.B   $9D, $9D, $9D, $9D, $9D, $00, $00, $00, $34, $35, $72, $74, $66, $67, $76, $62  ;$20
;          2    3    w    e    s    d    x    c    esc  1    SMC  q    QUL  a    QUL  y
  DATA.B   $32, $33, $77, $65, $73, $64, $78, $63, $1B, $31, $09, $71, $9F, $61, $9F, $79  ;$30
;          6    7    z    u    h    j    n    m    QUL  QUL  QUL  sp   QUL  O    SMC  .
  DATA.B   $36, $37, $7A, $75, $68, $6A, $6E, $6D, $9F, $9F, $9F, $20, $9F, $30, $9E, $2E  ;$40
;          8    9    i    o    k    l    ,    .    EST  EST  EST  EST  EST  SMC  EST  EST
  DATA.B   $38, $39, $69, $6F, $6B, $6C, $2C, $2E, $9D, $9D, $9D, $9D, $9D, $9E, $9F, $9F  ;$50
; LSB      0    1    2    3    4    5    6    7    8    9    A    B    C    D    E    F
  page
;
; THE CAPS/QUALIFIER FLAG TABLE
;         TABLE IS INDEXED BY KEYCODE.  EACH BYTE REPRESENTS THE ENTRY FOR
;         THE CORRESPONDING KEYCODE.
```

```
;
;  Used on Closure when a $9F is in the Keycode entry of the Shift or Regular
;       Table, process a Qualifier.  Also on Closure when the Caps Lock Key is
;       being pressed (CapsLock flag is set).
;  Used on Release when a $9E action code is in the Keycode entry of the Release
;       Table.
;
;  Each byte has 8 flags :
;       D7 = Caps lock flag : when set means this keycode generates a
;            shifted character when the Caps lock qualifier flag is set.
;       D6 = Qualifier has an ESC # sequence flag.  When set then must process
;            the keycode as a non-repeating ESC # sequence. Also has a Release
;            sequence.
;
;       D5 = Command -------
;       D4 = Alternate      !
;       D3 = Fast           !        This bit says which type of Qualifier
;       D2 = Caps lock      !______  key the Keycode represents.
;       D1 = Control        !
;       D0 = Shift   _______!
;
CQTABLE
                                                                               ;MSB
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$00
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$00,$00,$00,$00,$01 ;$10
  DATA.B  $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$80,$80,$80,$80,$80,$80 ;$20
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$00,$00,$00,$80,$04,$80,$01,$80 ;$30
  DATA.B  $00,$00,$80,$80,$80,$80,$80,$80,$02,$08,$60,$00,$10,$00,$00,$00 ;$40
  DATA.B  $00,$00,$80,$80,$80,$80,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00 ;$50
; LSB      0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F
  page
; ESCAPE SHARP(#) SEQUENCE TABLE
;        FORMAT :  (KEYCODE,FILLER,US/UC,SHIFT,COMMAND,C/S)
;
; Used on Closure when a $9D is in the Keycode entry of the Shift or
;       Regular Table.
; Used on Release when a $9D action code is in the Keycode entry of
;       the Release Table.  Releas keycode has high order bit set.
;
; The fill field is added to keep each record on an even byte boundary
; The other fields contain the character sequence to follow the ESCAPE #
;       characters:
;       US/UC = when the Shift and Command key are not pressed
;       SHIFT = when only the Shift key is still being pressed
;     COMMAND = when only the Command key is still being pressed
;         C/S = when the Shift and Command keys are still being pressed
;
ETABLE
;        KEYCODE FILL   US/UC  SHIFT  COMMAND   C/S
  DATA.B  $20,    0,    '00',  '0A',  '14',    '1E' ;FUNCTION KEY 1
  DATA.B  $21,    0,    '01',  '0B',  '15',    '1F' ;FUNCTION KEY 2
```

```
   DATA.B   $22,      0,      '02',    '0C',    '16',      '20' ;FUNCTION KEY 3
   DATA.B   $23,      0,      '03',    '0D',    '17',      '21' ;FUNCTION KEY 4
   DATA.B   $24,      0,      '04',    '0E',    '18',      '22' ;FUNCTION KEY 5
   DATA.B   $4A,      0,      'FF',    'FF',    'FF',      'FF' ;LEFT COMMAND (CLOSURE)
   DATA.B   $58,      0,      '05',    '0F',    '19',      '23' ;FUNCTION KEY 6
   DATA.B   $59,      0,      '06',    '10',    '1A',      '24' ;FUNCTION KEY 7
   DATA.B   $5A,      0,      '07',    '11',    '1B',      '25' ;FUNCTION KEY 8
   DATA.B   $5B,      0,      '08',    '12',    '1C',      '26' ;FUNCTION KEY 9
   DATA.B   $5C,      0,      '09',    '13',    '1D',      '27' ;FUNCTION KEY 10
   DATA.B   $CA,      0,      'FE',    'FE',    'FE',      'FE' ;LEFT COMMAND (RELEASE)
   page
;
; STANDARD MULTIPLE CHARACTER SEQUENCE TABLE
;          FORMAT :   (KEYCODE,LENGTH,CHARACTER_SEQUENCE)
;
; Used on Closure when a $9E is in the Keycode entry of the Shift or
;        Regular Table.
;
; The LENGTH field is the number of characters in the CHARACTER SEQUENCE field
; The CHARACTER SEQUENCE is the characters to return for the Keycode.
;
SMTABLE
;           KEYCODE LENGTH CHARACTER SEQUENCE
   DATA.B     $00,      2,      $1B,$43    ;CURSOR RIGHT
   DATA.B     $03,      2,      $1B,$48    ;HOME UP
   DATA.B     $07,      2,      $1B,$64    ;ENTER
   DATA.B     $08,      2,      $1B,$44    ;CURSOR LEFT
   DATA.B     $0B,      2,      $1B,$42    ;CURSOR DOWN
   DATA.B     $3A,      2,      $1B,$69    ;BACK TAB
   DATA.B     $5D,      2,      $1B,$41    ;CURSOR UP
   DATA.B     $4E,      2,      $30,$30    ;DOUBLE ZERO-( 00 KEY )
   DATA.B     $FF,      0                  ;NULL KEYCODE - END OF TABLE
   page
;
; RELEASE TABLE
;          FORMAT :   (KEYCODE,ACTION_CODE)
;
; Used on all Release keycodes.
;
; The action code describes the type of key:
;        9D = return an Escape Sharp Sequence for this keycode
;        9E = a Qualifier key
;all other = character code to return
;
RLTABLE
;           KEYCODE         ACTION CODE
   DATA.B     $1F,                $9E  ;RIGHT SHIFT
   DATA.B     $3C,                $9E  ;CAPS LOCK
   DATA.B     $3E,                $9E  ;LEFT SHIFT
   DATA.B     $48,                $9E  ;CONTROL
   DATA.B     $49,                $9E  ;FAST
```

```
  DATA.B     $4A,              $9E  ;LEFT COMMAND
  DATA.B     $4C,              $9E  ;ALTERNATE
  DATA.B     $FF,              $00  ;NULL KEYCODE - END OF TABLE
;
;
; BREAK KEYCODE TABLE
;        SINGLE BYTE TABLE.  THIS IS THE KEYCODE WHICH CAUSES START/STOP
;
; Used on all keycodes.
;
; The filler is to keep the file on an even byte boundary
;
BKEYCOD  DATA.B    $DF,0         ;BREAK CLOSURE KEYCODE,FILLER
;
LENGTH   EQU        %-VERSION     ;LENGTH OF DATA AFTER TRANSLATION TABLE
         END       START
```