OPERATING SYSTEM INTERFACE GUIDE

THE CORVUS CONCEPT

CORVUS SYSTEMS

DISCLAIMER OF ALL WARRANTIES & LIABILITIES

Corvus Systems, Inc. makes no warranties, either expressed or implied, with respect to this manual or with respect to the software described in this manual, its quality, performance, merchantability, or fitness for any particular purpose. Corvus Systems, Inc. software is sold or licensed "as is." The entire risk as to its quality or performance is with the buyer and not Corvus Systems, Inc., its distributor, or its retailer. The buyer assumes the entire cost of all necessary servicing, repair, or correction and any incidental or consequential damages. In no event will Corvus Systems, Inc. be liable for direct, indirect, incidental or consequential damages, even if Corvus Systems, Inc. has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of implied warranties or liabilities for incidental or consequential damages, so the above limitation may not apply to you.

Every effort has been made to insure that this manual accurately documents the operation and servicing of Corvus products. However, due to the ongoing modification and update of the software along with future products, Corvus Systems, Inc. cannot guarantee the accuracy of printed material after the date of publication, nor can Corvus Systems, Inc. accept responsibility for errors or omissions.

NOTICE

Corvus Systems, Inc. reserves the right to make changes in the product described in this manual at any time without notice. Revised manuals and update sheets will be published as needed and may be purchased by writing to:

Corvus Systems, Inc.
2029 O'Toole Avenue
San Jose, CA 95131

Telephone: (408) 946-7700
TWX 910-338-0226

FCC WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

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

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

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 file system provides a way to store data in named collections called files and a way to create, examine, remove, copy, and otherwise manipulate such files.
- the command interpreter, known as "the shell", provides the basic means of telling MERLIN what things it should do.
- the programming languages 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.

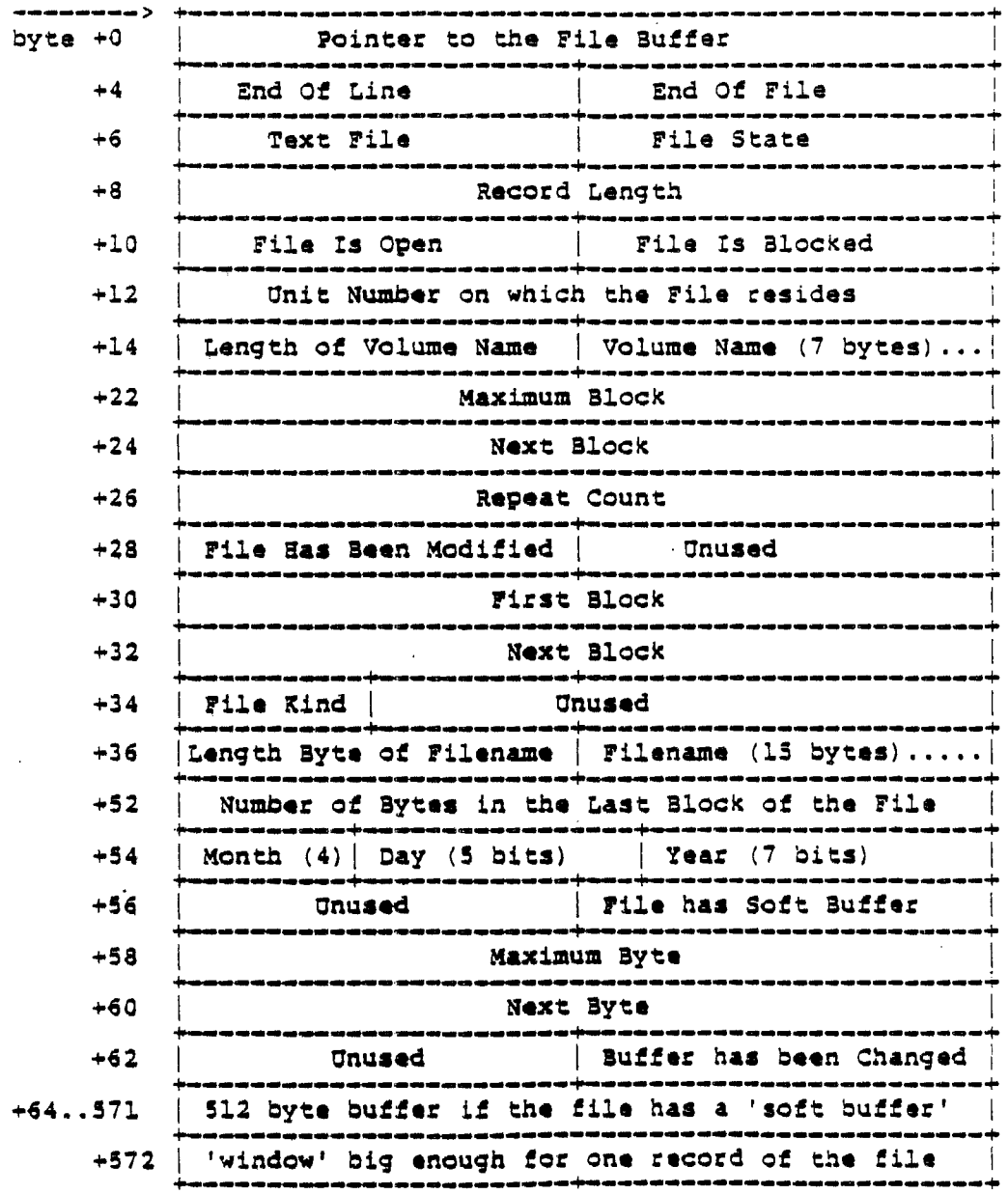
MAXIMUM BYTE is a word quantity that is the number of the last byte in the buffer. This is used when reading a file that has a partial last block or when writing any file.

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.

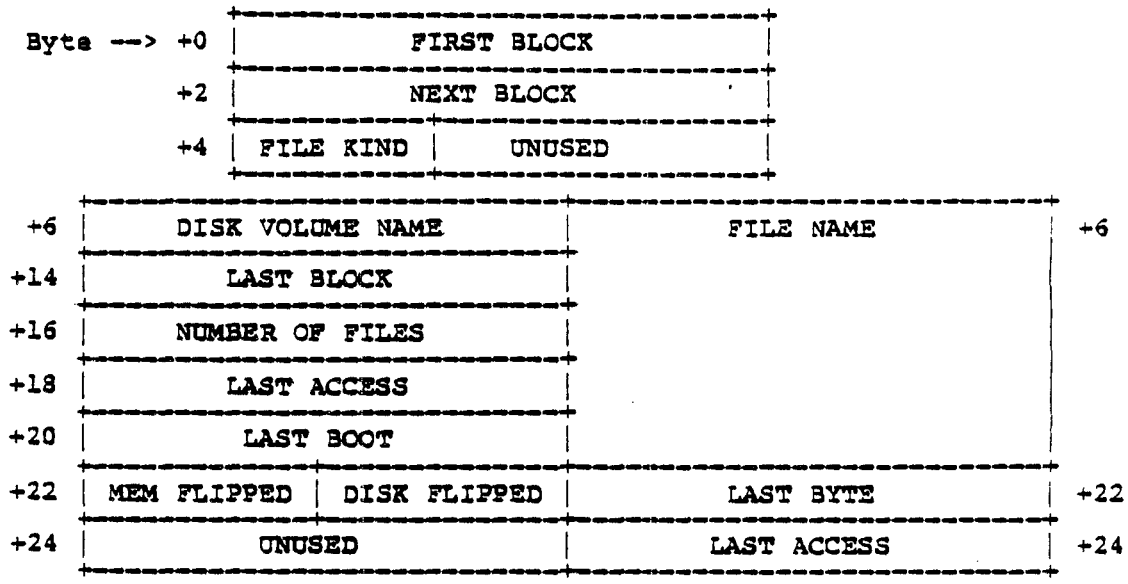


Figure 2-3
Layout of a Directory Entry

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.

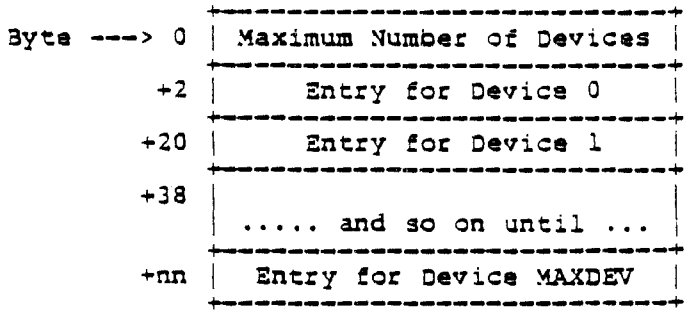


Figure 2-4
Overall Layout of the Device Table

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.

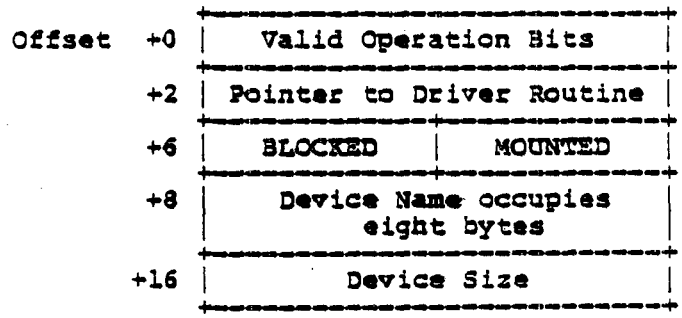


Figure 2-5
Individual Device Table Entries

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

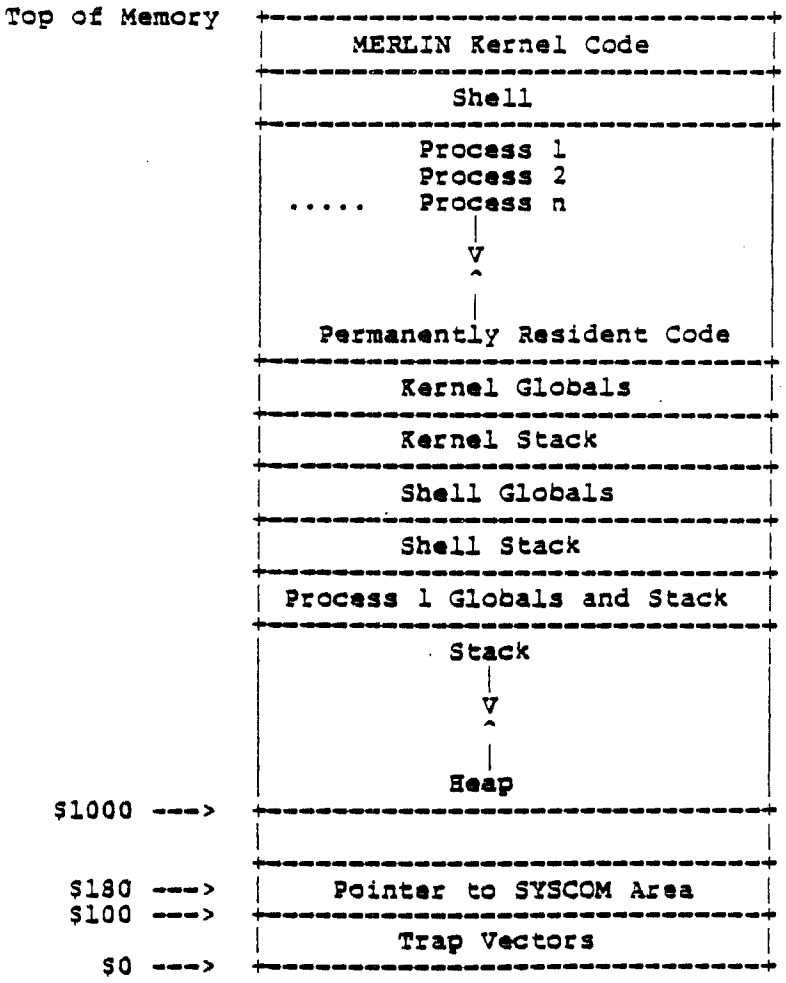


Figure 2-6
Memory Layout in MERLIN

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.

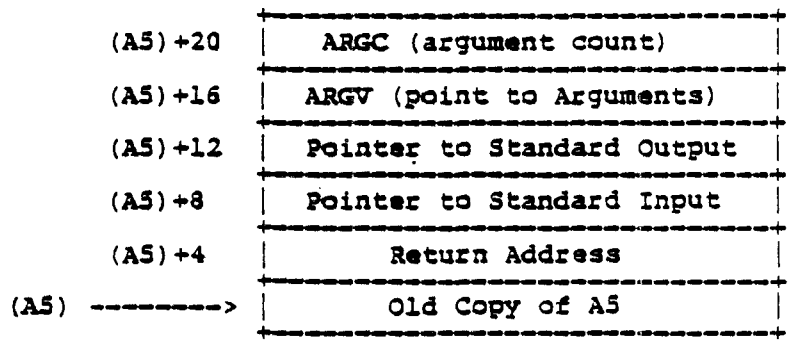


Figure 2-7
Environment of a Running Program

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:

pointer to FIB 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:

Pointer to Filename
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.

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

New File Indicator
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 mode parameter described below. Parameters are:

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

Mode — a word quantity indicating the disposition of the file after it is closed. The modes are:

0	normal - if the file is an old file - it existed prior to this program run, it is saved (retained) in the file system. If the file is a new file - created during this program run, it is deleted or purged from the file system.
1	lock - makes a file permanent in the file system, regardless of any conditions mentioned in case (0) above.
2	purge - 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:

Pointer to FIB — 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:

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

Character — to be written is a byte.

Size — a word quantity representing a field width. If size 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:

Pointer to Storage
: 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.

3.4 GETDIR - Read a Directory

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

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

Pointer to Directory
: a long word pointer to an area of memory large enough to receive an entire directory.

Device Blocked indicator
: is a long word pointer to a Boolean quantity which is set true if the device is blocked device.

Device Number
: 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 Direntry 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

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".

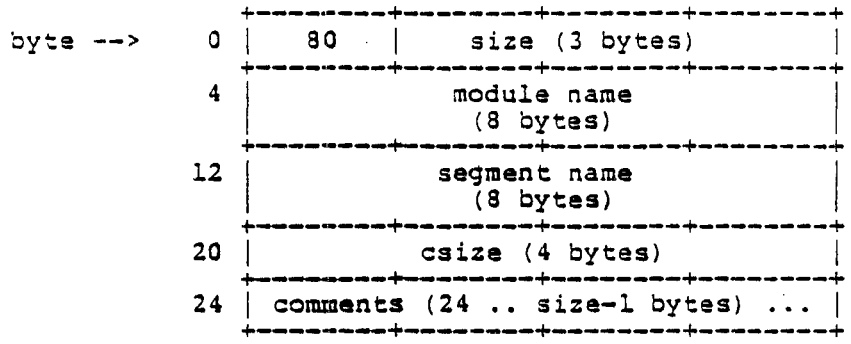
```
<Link File>     ::= <Module File>
                ::=   <Library File>
```

```
                ::=     <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

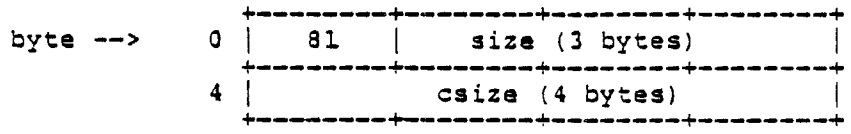
All Linker and Librarian object-code blocks start with a single "identifier byte". This block identifier takes values from 80 (base 16) upwards. The choice of values greater than 80 (base 16) is an attempt to minimise the probability that a regular ASCII text file is mistaken for the start of an object-code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

4.3.1 80 - Module Name Block



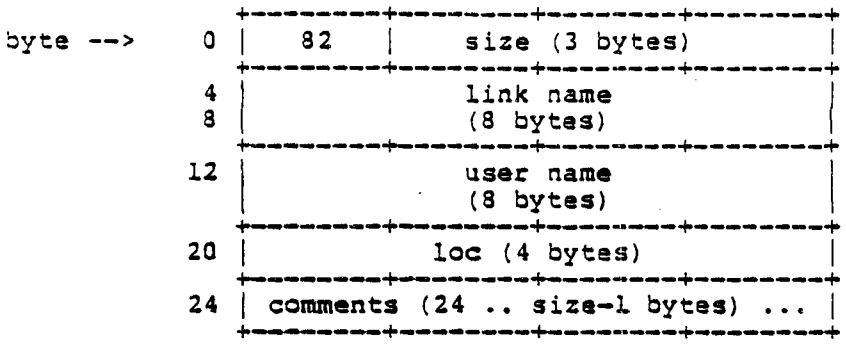
80	Hexadecimal 80 indicates a Module Name Block.
size	Number of bytes in this block.
module name	Blank padded ASCII name of module.
segment name	ASCII name of segment in which this module will reside.
csize	Number of bytes in the code block for this module.
comments	Arbitrary information - ignored by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

4.3.2 81 - End Block



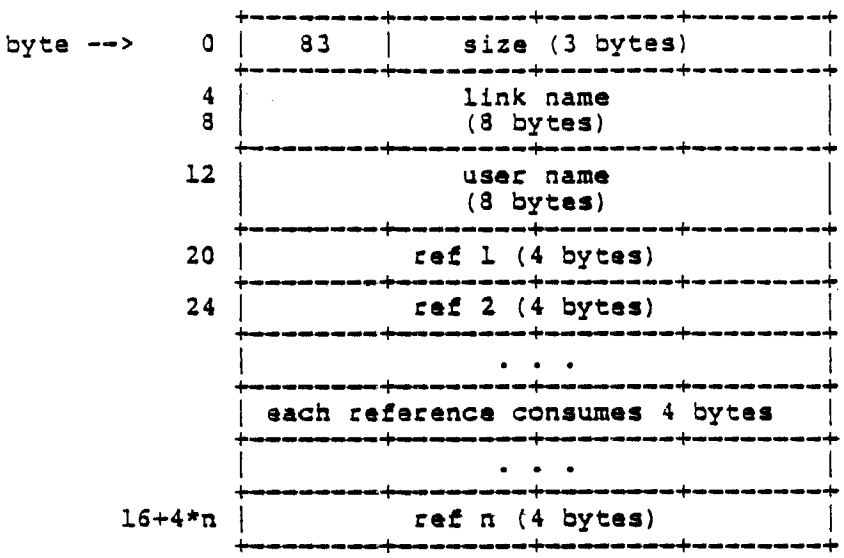
81	Hexadecimal 81 indicates this is an End Block.
size	Number of bytes in this block - it is always 000008.
csize	Number of bytes in the code block for this module.

4.3.3 82 - Entry Point Block



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



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.

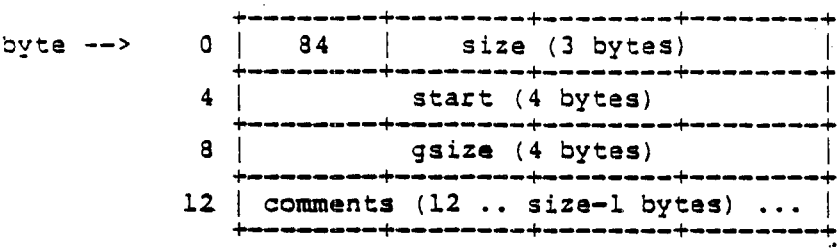
ref 1 Location of first reference relative to this module.

ref 2 Location of second reference relative to this module.

. . . Other references.

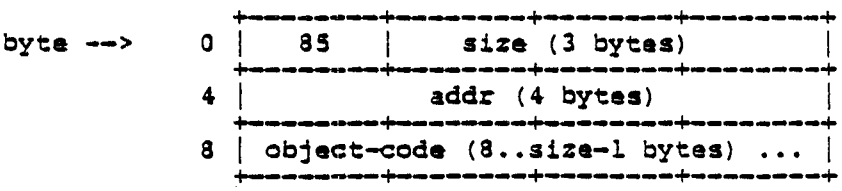
ref n Location of last reference relative to this module.

4.3.5 84 - Starting Address Block



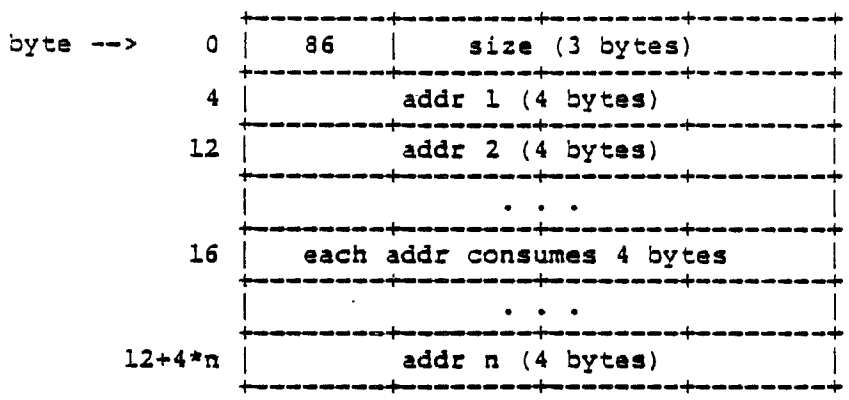
84	Hexadecimal 84 indicates this is a Starting Address Block.
size	Number of bytes in this block.
start	Starting address relative to this module.
gsize	Number of bytes in the global data area.
comments	Arbitrary information - ignored by the Linker.

4.3.6 85 - Code Block



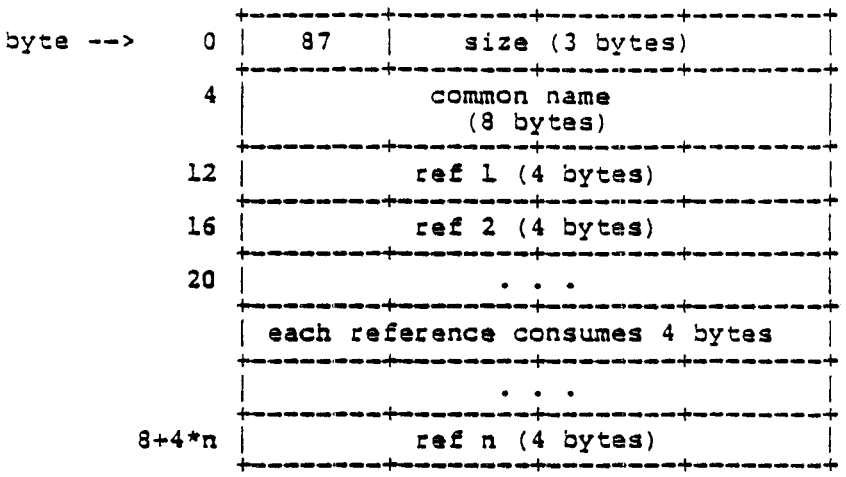
85	Hexadecimal 85 indicates this is a Code Block.
size	Number of bytes in this block.
addr	Module-relative address of first code byte.
object-code	The object-code - always an even number of bytes.

4.3.7 86 - 32-Bit Relocation Block



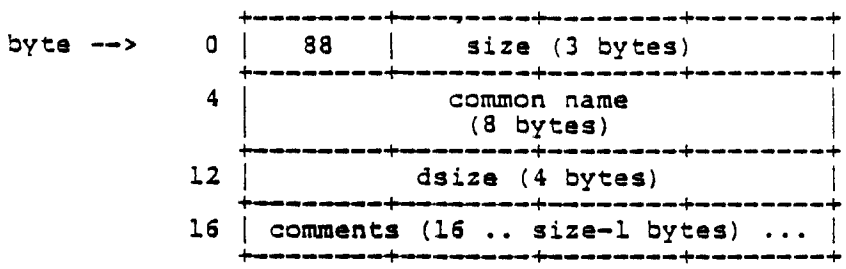
86	Hexadecimal 86 indicates this is a 32-bit Relocation Block.
size	Number of bytes in this block.
addr 1	Location of first address to relocate.
addr 2	Location of second address to relocate.
. . .	Locations of other addresses to relocate.
addr n	Location of last address to relocate.

4.3.8 87 - Common Block Reference



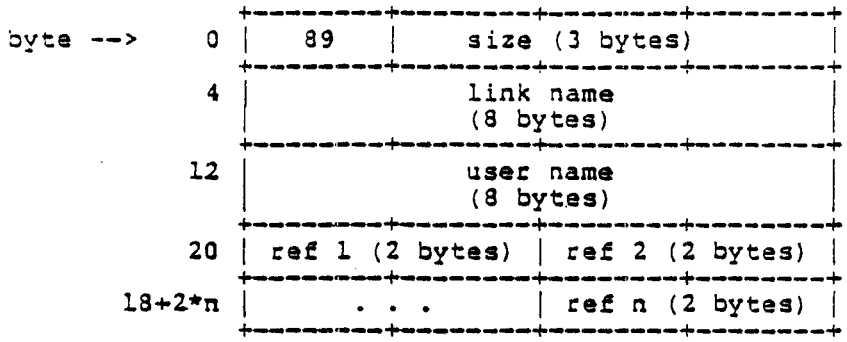
87	Hexadecimal 87 indicates this is a Common Block Reference.
size	Number of bytes in this block.
common name	Blank padded ASCII common block name.
ref 1	Location of first reference relative to this module.
ref 2	Location of second reference relative to this module.
. . .	Other references relative to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

4.3.9 88 - Common Block Definition



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.

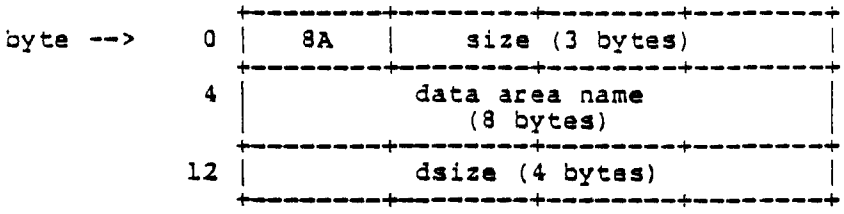
4.3.10 89 - Short External Reference Block



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



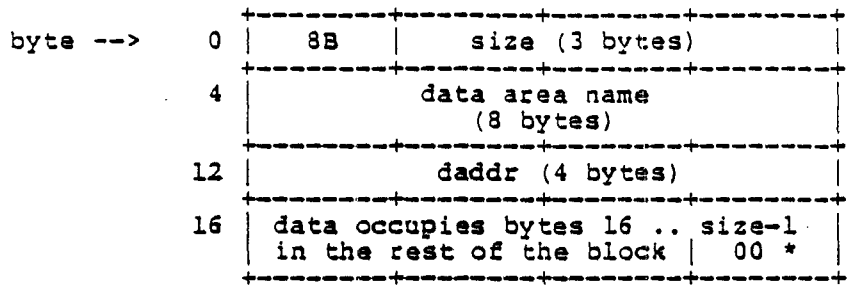
8A — Hexadecimal 8A indicates this is a FORTRAN Data Area Definition Block.

size — Number of bytes in this block.

data area name — Blank padded ASCII name of FORTRAN fixed data area.

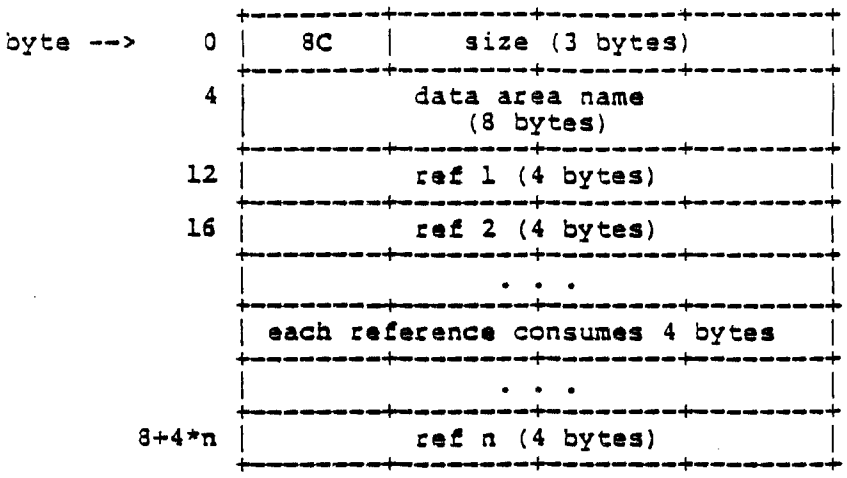
dsize — Size of this data area.

4.3.12 8B - FORTRAN Data Area Initialization Block



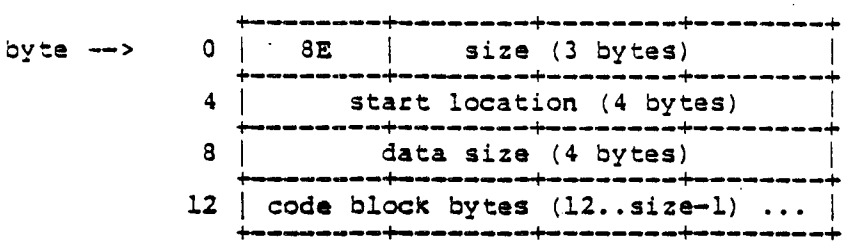
8B	Hexadecimal 8B indicates this is a FORTRAN Data Area Initialization Block.
size	Number of bytes in this block.
data area name	Blank padded ASCII name of FORTRAN fixed data area.
daddr	Starting address for this data.
data	The initialization data.
00 *	If the size of the data block is odd, there is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



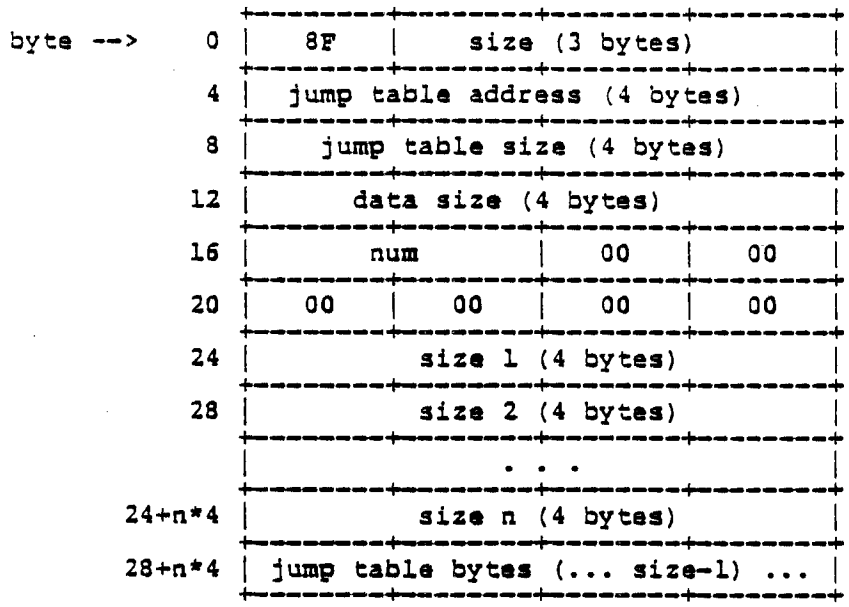
8C	Hexadecimal 8C indicates this is a FORTRAN Data Area Reference Block.
size	Number of bytes in this block.
data area name	Blank padded ASCII name of FORTRAN fixed data area.
ref 1	Location of first reference.
ref 2	Location of first reference.
. . .	Location of other references.
ref n	Location of last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

4.3.14 8E - Quick Load Executable Block



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



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.

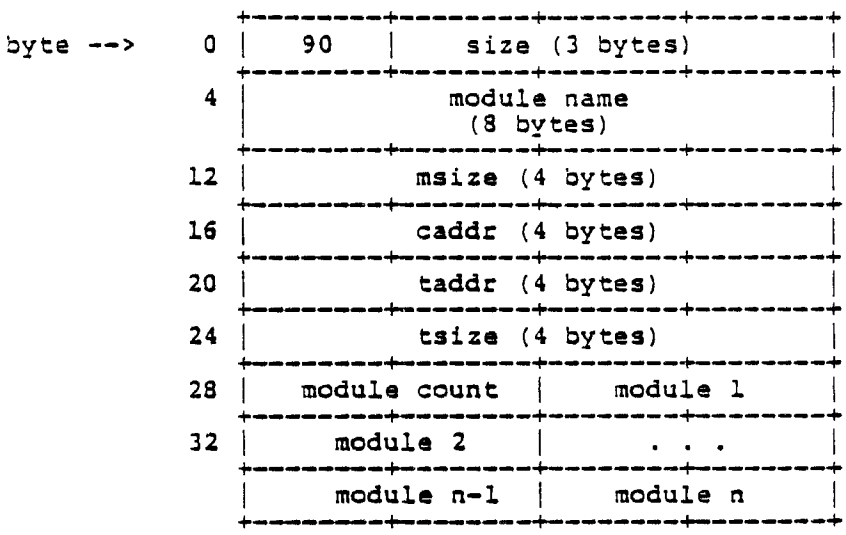
num Number of FORTRAN Data Areas.

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".

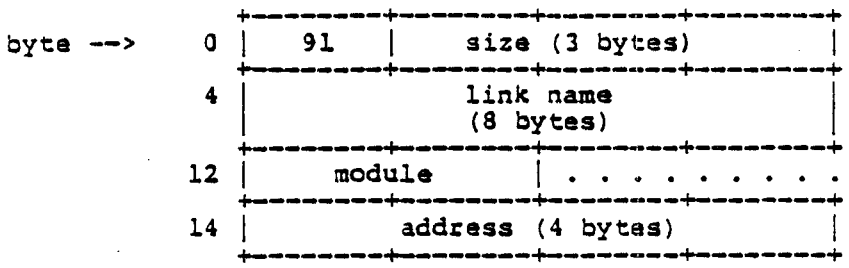
4.3.16 90 - Library Module Block



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.

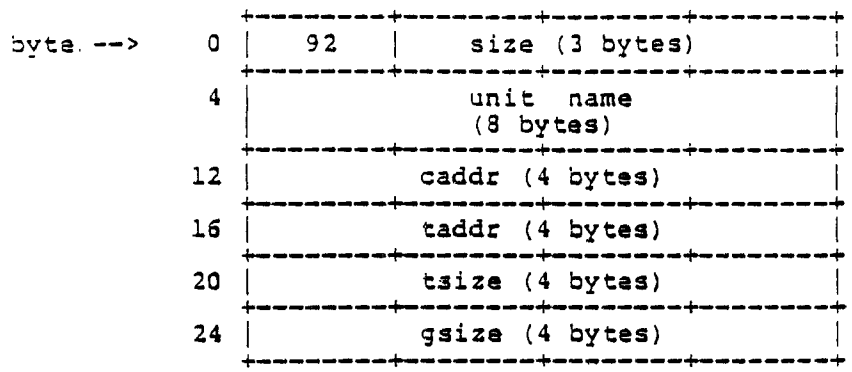
. . .	Numbers of other modules referenced.
module n	Number of the last module referenced.

4.3.17 91 - Library Entry Block



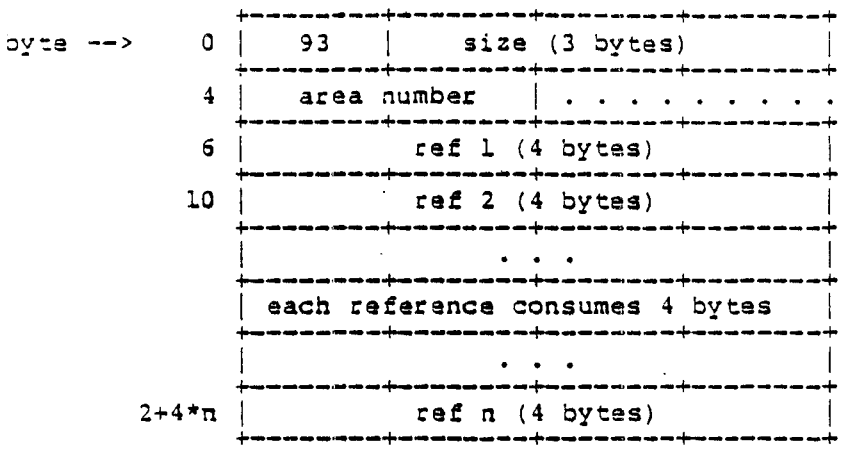
byte				
0	91	size (3 bytes)		
4	link name (8 bytes)			
12	module	. . . . .		
14	address (4 bytes)			

4.3.18 92 - Unit Block



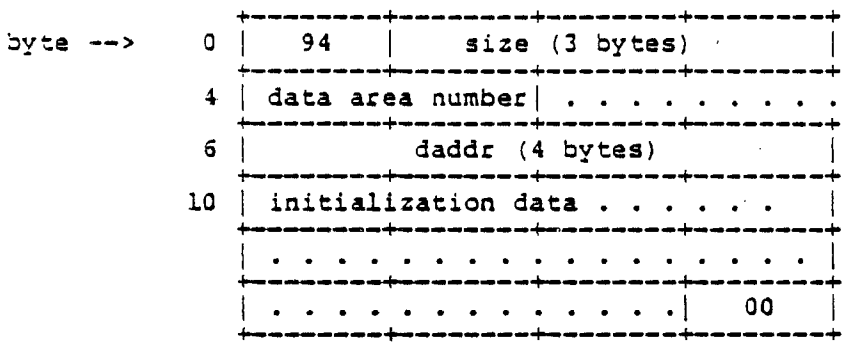
92	Hexadecimal 92 indicates that this is a Unit Block.
size	Number of bytes in this block - always 00001C.
unit name	Name of this unit.
caddr	Disk address of module.
taddr	Disk address of text block.
tsize	Size of text block.
gsize	Number of bytes of globals in this unit.

4.3.19 93 - FORTRAN Executable Data Area Reference Block



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



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.

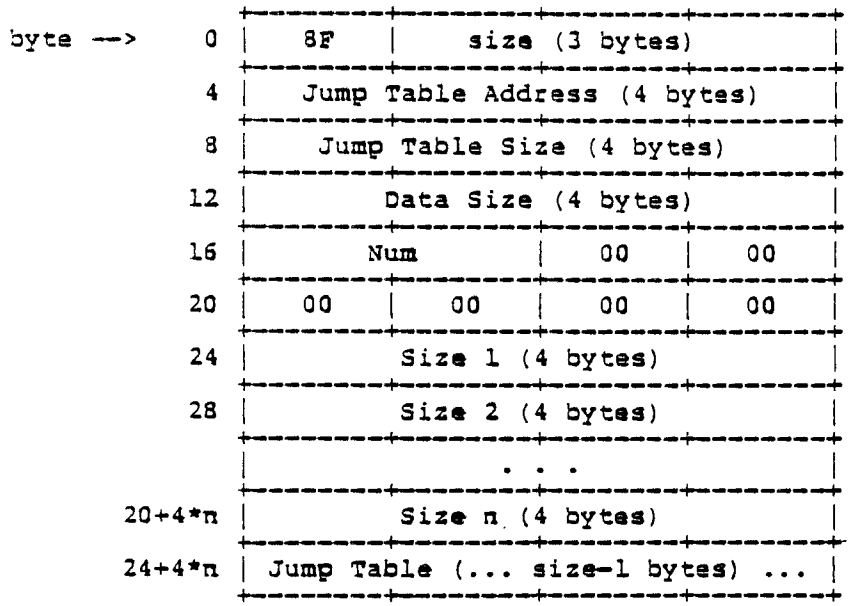
initialization data
The data to fill the block with.

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



8F — Hexadecimal 8F indicates this is an Executable Block Definition.

size — Number of bytes in this block.

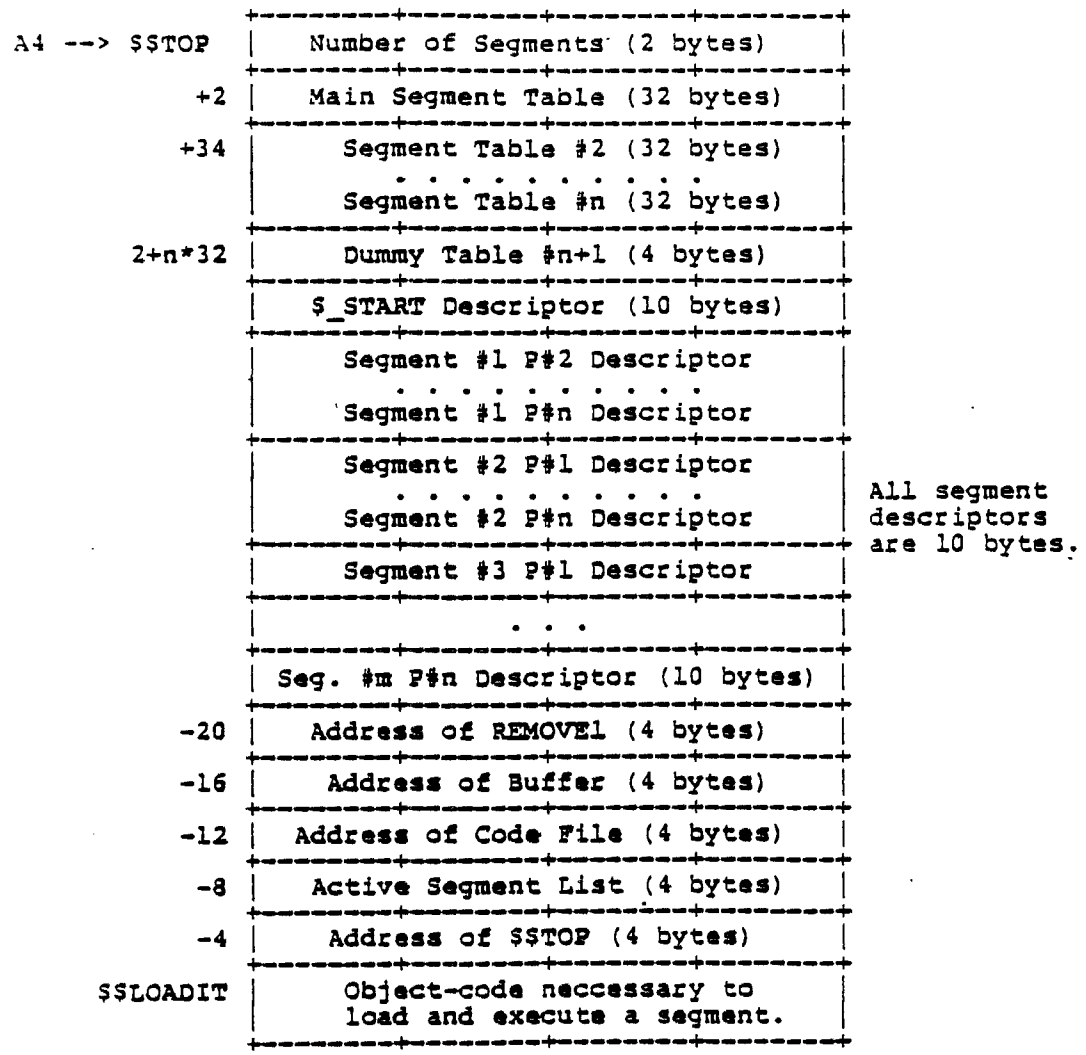
jump table address — Absolute load address of jump table.

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.

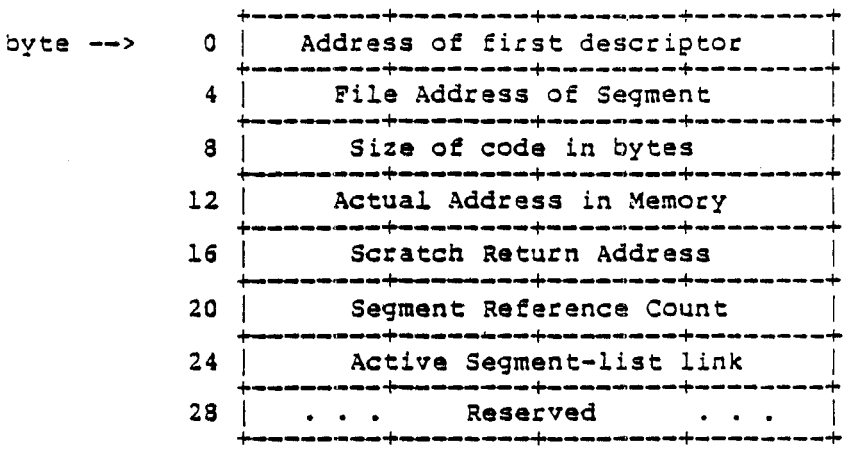
If any FORTRAN Executable Data Area Initialization Blocks are present, they must immediately follow the executable block.

4.4.2 Format of the Jump Table



4.4.3 Layout of a Segment Table

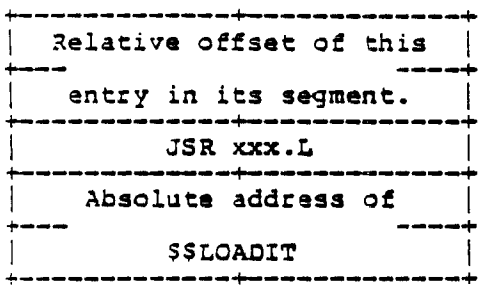
A Segment Table consists of eight 32-bit values:



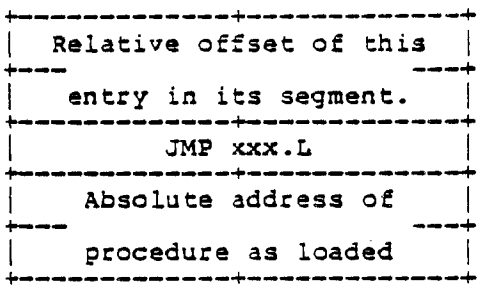
4.4.4 Layout of Descriptors

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:



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
```



```
;
; DomRD - Read character(s) from DataComm
;
DomRD                                   ;
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

;
; DomWR - Write character(s) to DataComm
;
DomWR                                   ;
CwrLoop subq.w   #1,d2                  ; More to write?
        bmi.s    CwrExit                ; No, return
        .... remaining logic similar
           .... to DomRD 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

```
                        1* ; File: cclib.bit.text
                        2* ; Date. 13-May-82
                        3*
                        4* ;
                        5* ; Corvus CONCEPT bit manipulation functions
                        6* ;
                        7*
                        8*         GLOBAL BITFLIP,BITSET,BITCLEAR,BITTEST,SHIFTRT,SHIFTLT,MAKEBYTE
                        9*
                       10* ;
                       11* ; Function BitFlip (data,bitnum. integer). integer,
                       12* ;
0000  205F             13* BITFLIP MOVE.L  (SP)+,A0        , A0 = return address
0002  4C9F  0003       14*         MOVEM.W (SP)+,D0-D1     , D0 = bit nmbr, D1 = data word
0006  0141             15*         BCHG    D0,D1           ; flip the bit
0008  3E81             16*         MOVE.W  D1,(SP)         , place changed word on stack
000A  4ED0             17*         JMP     (A0)            ; return to Pascal
                       18*
                       19* ;
                       20* ; Function BitSet (data,bitnum. integer): integer,
                       21* ;
000C  205F             22* BITSET  MOVE.L  (SP)+,A0        ; A0 = return address
000E  4C9F  0003       23*         MOVEM.W (SP)+,D0-D1     ; D0 = bit nmbr, D1 = data word
0012  01C1             24*         BSET    D0,D1           ; set the bit
0014  3E81             25*         MOVE.W  D1,(SP)         ; place changed word on stack
0016  4ED0             26*         JMP     (A0)            ; return to Pascal
                       27*
                       28* ;
                       29* ; Function BitClear (data,bitnum: integer): integer,
                       30* ;
0018                   31* BITCLEAR
0018  205F             32*         MOVE.L  (SP)+,A0        ; A0 = return address
001A  4C9F  0003       33*         MOVEM.W (SP)+,D0-D1     ; D0 = bit nmbr, D1 = data word
001E  0181             34*         BCLR    D0,D1           ; clear the bit
0020  3E81             35*         MOVE.W  D1,(SP)         ; place changed word on stack
0022  4ED0             36*         JMP     (A0)            , return to Pascal
                       37*
                       38* ;
                       39* ; Function BitTest (data,bitnum: integer): boolean,
                       40* ;
0024  205F             41* BITTEST MOVE.L  (SP)+,A0        ; A0 = return address
0026  4C9F  0003       42*         MOVEM.W (SP)+,D0-D1     , D0 = bit nmbr, D1 = data word
002A  4257             43*         CLR.W   (SP)            , assume false = 0
002C  0101             44*         BTST    D0,D1           ; test the bit
002E  6704             45*         BOFF.S  BTX             ,
0030  1EBC  0001       46*         MOVE.B  #1,(SP)         , bit is on. return true
0034  4ED0             47* BTX     JMP     (A0)            ; return to Pascal
                       48*
```

```
                        50* ;
                        51* ; Function ShiftRt (data: integer): integer;
                        52* ;
0036  205F              53* SHIFTRT MOVE.L  (SP)+,A0        ; A0 = return address
0038  301F              54*         MOVE.W  (SP)+,D0        ; D0 = word to be shifted
003A  E248              55*         LSR.W   #1,D0           ; shift it right
003C  3E80              56*         MOVE.W  D0,(SP)         ; push result on stack
003E  4ED0              57*         JMP     (A0)            ; return to Pascal
                        58*
                        59* ;
                        60* ; Function ShiftLt (data: integer): integer;
                        61* ;
0040  205F              62* SHIFTLT MOVE.L  (SP)+,A0        ; A0 = return address
0042  301F              63*         MOVE.W  (SP)+,D0        ; D0 = word to be shifted
0044  E348              64*         LSL.W   #1,D0           ; shift it left
0046  3E80              65*         MOVE.W  D0,(SP)         ; push result on stack
0048  4ED0              66*         JMP     (A0)            ; return to Pascal
                        67*
                        68* ;
                        69* ; Function MakeByte (n: integer): byte;
                        70* ;
004A                    71* MAKEBYTE
004A  205F              72*         MOVE.L  (SP)+,A0
004C  301F              73*         MOVE.W  (SP)+,D0        ; get n
004E  1E80              74*         MOVE.B  D0,(SP)         ; return function value
0050  4ED0              75*         JMP     (A0)            ; return to Pascal
                        76*
                        77*         END

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

0 errors.  78 lines.
```

```
1* ; File: cclib.asm.text
2* , Date. 06-Oct-82
3*
4* ;
5* ; Corvus CONCEPT operating system interface
6* ,
7*
8*          IDENT   CCLIBASM
9*          GLOBAL  OSactSlt,OSactSrv,OSaltSlt,OSaltSrv,OSsltType
10*         GLOBAL  OSmaxDev,OSdispDv,OSkybdDv,OStimDv
11*         GLOBAL  OSomniDv,OSdomZDv,OSdomlDv,OSsltDv
12*         GLOBAL  OSextCRT,pOSuserID,pOScurWnd,pOSsysWnd,pOSdevNam,pOSdate
13*         GLOBAL  xGetDir,xPutDir
14*
15*         include '/ccos/os.gbl.asm.text'
```

```
                    17* ,
                    18* ; File: os.gbl.asm.text
                    19* , Date: 10-Aug-83
                    20* ;
                    21* ;
                    22* , Corvus CONCEPT operating system data structure equates
                    23* ;
                    24*
                    25* ;
                    26* ; Additional Corvus CONCEPT I/O result codes
                    27* ,
00000003            28* IOEioreq equ   3       ; Invalid I/O request
                    29*                       ;
00000015            30* IOEnotrn equ   21      ; Transporter not ready
00000016            31* IOEtimot equ   22      ; Timed out waiting for Omninet event
00000017            32* IOEnobuf equ   23      ; Read without a valid write buffer
                    33*                       ;
00000020            34* IOEwndfn equ   32      ; Invalid window function
00000021            35* IOEwndbe equ   33      , Window create boundary
00000022            36* IOEwndcs equ   34      , Invalid character set
00000023            37* IOEwndde equ   35      , Delete current window
00000024            38* IOEwndds equ   36      ; Delete system window
00000025            39* IOEwndiw equ   37      ; Inactive window
00000026            40* IOEwndwr equ   38      ; Invalid window record
00000027            41* IOEwndwn equ   39      , Invalid system window number
                    42*                       ;
00000028            43* IOEnodsp equ   40      , Display driver not available
00000029            44* IOEnokyb equ   41      , Keyboard driver not available
0000002A            45* IOEnotim equ   42      , Timer driver not available
0000002B            46* IOEncomm equ   43      ; OMNINET driver not available
0000002C            47* IOEnoprt equ   44      ; Printer driver not available
0000002D            48* IOEnfdrv equ   45      , No floppy drive at slot
                    49*                       ;
00000032            50* IOEtblid equ   50      ; Invalid table entry ID
00000033            51* IOEtblfl equ   51      , Table full
00000034            52* IOEtblin equ   52      , Table entry in use
00000035            53* IOEkybte equ   53      ; Keyboard transmission error
00000036            54* IOEuiopm equ   54      ; Invalid unit I/O parameter
00000037            55* IOEprmln equ   55      ; Invalid parameter block length
00000038            56* IOEfncod equ   56      , Invalid function code
00000039            57* IOEclkmf equ   57      ; Clock (hardware) malfunction
                    58*
                    59* ;
                    60* ; System Common Pointer
                    61* ;
00000100            62* pSysCom    equ $0100   ,pointer to address of SYSCOM
00000104            63* SysKybdFlg equ $0104   ;keyboard control flags
00000106            64* SysByteScn equ $0106   ;display driver - bytes per scan line
                    65*
                    66* ;
                    67* ; System Common Equates
                    68* ;
00000000            69* SCiorslt equ   0       ;word - I/O result
00000002            70* SCprocno equ   2       ;word - current process number
```

```
00000004          71* SCfreehp equ   4      ;lint - free heap pointer
00000008          72* SCjtable equ   8      ;lint - jump table pointer
0000000C          73* SCsysout equ   12     ;lint - default output file pointer
00000010          74* SCsysin  equ   16     ;lint - default input file pointer
00000014          75* SCdevtab equ   20     ;lint - device (unit) table pointer
00000018          76* SCdirnam equ   24     ;lint - directory name string pointer
0000001C          77* SCutable equ   28     ;lint - user table pointer
00000020          78* SCtoday  equ   32     ;word - system date
00000022          79* SCcodejt equ   34     ;lint - code jump table pointer
00000026          80* SCnxtpro equ   38     ;word - next process number
00000028          81* SCnumpro equ   40     ;word - number of processes
0000002A          82* SCprotbl equ   42     ;lint - process table pointer
0000002E          83* SCbootnm equ   46     ;lint - boot device name pointer
00000032          84* SCmemmap equ   50     ;lint - memory map pointer
00000036          85* SCbootdv equ   54     ;word - boot device number
                  86*                       ;
                  87*                       ; CONCEPT additions
                  88* ,        equ   56     ;word - unused
                  89* ;        equ   58     ;word - unused
0000003C          90* SCslttbl equ   60     ;lint - slot table pointer
00000040          91* SCrootw  equ   64     ;lint - root window record pointer
00000044          92* SCcurrw  equ   68     ;lint - current window record pointer
00000048          93* SCcurrk  equ   72     ;lint - current keyboard record pointer
0000004C          94* SCuserid equ   76     ;word - Constellation user ID
0000004E          95* SCvrsnbr equ   78     ;lint - current version number string pointer
00000052          96* SCvrsdat equ   82     ;lint - current version date string pointer
00000056          97* SCwndtbl equ   86     ;lint - window table pointer
0000005A          98* SCsusinh equ   90     ;word - suspend inhibit count
0000005C          99* SCsusreq equ   92     ;word - suspend request if non-zero
                 100*
```

```
                    102*
                    103* , System Vector Equates
                    104* ;
00000000            105* SVuwrite equ     0*4     ;unit write
00000004            106* SVuread  equ     1*4     ;unit read
00000008            107* SVuclear equ     2*4     ;unit clear
0000000C            108* SVubusy  equ     3*4     ;unit busy
00000010            109* SVput    equ     4*4     ;put
00000014            110* SVget    equ     5*4     ;get
00000018            111* SVinit   equ     6*4     ;init
0000001C            112* SVopen   equ     7*4     ;open
00000020            113* SVclose  equ     8*4     ;close
00000024            114* SVwrchar equ     9*4     ;writechar
00000028            115* SVrdchar equ     10*4    ;readchar
0000002C            116* SVblkio  equ     11*4    ;blockio
00000030            117* SVseek   equ     12*4    ;seek
00000034            118* SVnew    equ     13*4    ;new
00000038            119* SVdspose equ     14*4    ;dispose
0000003C            120* SVmark   equ     15*4    ;mark
00000040            121* SVrlease equ     16*4    ;release
00000044            122* SVmavail equ     17*4    ;memory available
00000048            123* SVgetdir equ     18*4    ;get directory
00000060            124* SVcrkpth equ     24*4    ;crack path name
00000064            125* SVustat  equ     25*4    ;unit status
0000007C            126* SVcli    equ     31*4    ;command line interpreter
00000080            127* SVgetvnm equ     32*4    ;get volume names
00000084            128* SVvaldir equ     33*4    ;check valid directory
00000088            129* SVflpdir equ     34*4    ;flip directory
0000008C            130* SVschdir equ     35*4    ;search directory
00000090            131* SVdelent equ     36*4    ;delete directory entry
00000094            132* SVputdir equ     37*4    ;write directory
00000098            133* SVuinstl equ     38*4    ;unit install
                    134*
                    135* ,
                    136* , Memory Map Equates
                    137* ;
00000000            138* MMlodta  equ     0       ;lint - low data pointer
00000004            139* MMhidta  equ     4       ;lint - high data pointer
00000008            140* MMlocod  equ     8       ;lint - low code pointer
0000000C            141* MMhicod  equ     12      ;lint - high code pointer
00000010            142* MMbtsw   equ     16      ;word - boot switches
00000012            143* MMbtdev  equ     18      ;word - boot device number
00000014            144* MMbtslt  equ     20      ;word - boot slot number
00000016            145* MMbtsrv  equ     22      ;word - boot server number
00000018            146* MMbtdrv  equ     24      ;word - boot drive number
0000001A            147* MMbtblk  equ     26      ;word - boot volume block number
                    148*
```

```
                    150* ;
                    151* ; Unit Table Equates
                    152* ;
00000002            153* UTiodrv equ     2       ,lint - I/O driver pointer
00000006            154* UTblf  equ     6       ;bool - blocked device flag
00000007            155* UTmtd  equ     7       ,bool - mounted device flag
00000008            156* UTdid  equ     8       ,str7 - device ID
00000010            157* UTsiz  equ     16      ;lint - device size
00000014            158* UTslt  equ     20      ;byte - device slot
00000015            159* UTsrv  equ     21      ;byte - device server
00000016            160* UTdrv  equ     22      ,byte - disk drive nmbr
00000017            161* UTtyp  equ     23      ;byte - disk drive type
00000018            162* UTspt  equ     24      ;byte - sectors per track
00000019            163* UTtps  equ     25      ,byte - tracks per side
0000001A            164* UTro   equ     26      ;bool - device read only
                    165* ;      equ     27      ,byte - ... unused
0000001C            166* UTblk  equ     28      ,lint - disk base block
00000020            167* UTlen  equ     32      ,       entry length
                    168*
                    169* ;
                    170* , Slot Table Equates
                    171* ,
00000000            172* STbtslt  equ    0       ,boot slot number
00000002            173* STbtsrv  equ    2       ,boot server number
00000004            174* STacslt  equ    4       ,active slot number
00000006            175* STacsrv  equ    6       ,active server number
00000008            176* STalslt  equ    8       ;alternate slot number
0000000A            177* STalsrv  equ    10      ;alternate server number
0000000C            178* STinfo   equ    12      ;array [1..5] of ....
                    179*                        ;
00000000            180* STnmbr   equ    0       ;   slot number (1-5)
00000001            181* STtype   equ    1       ,   device type (slottypes)
00000002            182* STndrv   equ    2       ,   number of drives
00000004            183* STinfoL  equ    4       ,   device info length
                    184*
```

```
                    184*
                    185* ; Character Set Record Equates
                    186*
00000000            187* CStblloc equ   0       ;character set data pointer
00000004            188* CSlpch   equ   4       ;scanlines per character (assume wide)
00000006            189* CSbpch   equ   6       ;bits per character (vertical height)
00000008            190* CSfrstch equ   8       ;first character code - ascii
0000000A            191* CSlastch equ  10       ;last character code - ascii
0000000C            192* CSmask   equ  12       ;mask used in positioning cells
00000010            193* CSattr1  equ  16       ;attributes
                    194*                        ;  bit 0 = 1 - vertical orientation
00000011            195* CSattr2  equ  17       ;currently unused
                    196*
                    197*
                    198* ; Window Record Equates
                    199*
00000000            200* WRcharpt equ   0       ;character set pointer
00000004            201* WRhomept equ   4       ;home (upper left) pointer
00000008            202* WRcuradr equ   8       ;current location pointer
0000000C            203* WRhomeof equ  12       ;bit offset of home location
0000000E            204* WRbasex  equ  14       ;home x value, relative to root window
00000010            205* WRbasey  equ  16       ;home y value, relative to root window
00000012            206* WRlngthx equ  18       ;maximum x value, relative to window (bits)
00000014            207* WRlngthy equ  20       ;maximum y value, relative to window (bits)
00000016            208* WRcursx  equ  22       ;current x value (bits)
00000018            209* WRcursy  equ  24       ;current y value (bits)
0000001A            210* WRbitofs equ  26       ;bit offset of current address
0000001C            211* WRgrorgx equ  28       ;graphics - origin x (bits relative to home loc)
0000001E            212* WRgrorgy equ  30       ;graphics - origin y (bits relative to home loc)
00000020            213* WRattr1  equ  32       ;attributes
                    214*                        ;
00000000            215* inverse  equ   0       ;   inverse video mode
00000001            216* undscr   equ   1       ;   underscore mode
00000002            217* insmod   equ   2       ;   insert mode
00000003            218* viddeflt equ   3       ;   0 = W on B,       1 = B on W
00000004            219* noautolf equ   4       ;   0 = auto LF w/CR, 1 = no auto LF
00000005            220* syswin   equ   5       ;   system defined window
00000006            221* active   equ   6       ;   active window
00000007            222* suspend  equ   7       ;   suspended window
                    223*                        ;
00000021            224* WRattr2  equ  33       ;attributes
                    225*                        ;
00000000            226* vert     equ   0       ;   1 = vertical,   0 = horizontal screen
00000001            227* graphic  equ   1       ;   1 = graphics,   0 = character mode
00000002            228* curson   equ   2       ;   1 = cursor on,  0 = cursor off
00000003            229* invcurs  equ   3       ;   1 = inverse,    0 = underline cursor
00000004            230* wrapon   equ   4       ;   1 = wrap,       0 = clip at eoln
00000005            231* noscroll equ   5       ;   1 = no scroll,  0 = scroll
00000006            232* clrsc    equ   6       ;   1 = paging mode
00000007            233* vidset   equ   7       ;   1 = inverse     0 = normal
                    234*                        ;
00000022            235* WRstate  equ  34       ;used for decoding escape sequences
00000023            236* WRrcdlen equ  35       ;window description record length
                    237*                        ;
```

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

```
                        143*
                        144* ,
                        145* , OSACTSLT - Get active slot function
                        146* ;
                        147* ; FUNCTION OSactSlt: integer;
                        148* ;
0000                    149* OSactSlt
0000  2278  0180        150*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0004  2269  003C        151*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0008  3F69  0004  0004  152*        move.w  STacslt(a1),4(sp)       ;Get active slot from slot table
000E  4E75              153*        rts                             ;Return
                        154*
                        155* ;
                        156* , OSACTSRV - Get active server function
                        157* ;
                        158* ; FUNCTION OSactSrv: integer;
                        159* ;
0010                    160* OSactSrv
0010  2278  0180        161*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0014  2269  003C        162*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0018  3F69  0006  0004  163*        move.w  STacsrv(a1),4(sp)       ;Get active server from slot table
001E  4E75              164*        rts                             ;Return
                        165*
                        166* ;
                        167* ; OSALTSLT - Get alternate slot function
                        168* ;
                        169* ; FUNCTION OSaltSlt: integer;
                        170* ;
0020                    171* OSaltSlt
0020  2278  0180        172*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0024  2269  003C        173*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0028  3F69  0008  0004  174*        move.w  STalslt(a1),4(sp)       ;Get alternate slot from slot table
002E  4E75              175*        rts                             ;Return
                        176*
                        177* ;
                        178* ; OSALTSRV - Get alternate server function
                        179* ;
                        180* ; FUNCTION OSaltSrv: integer;
                        181* ;
0030                    182* OSaltSrv
0030  2278  0180        183*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0034  2269  003C        184*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0038  3F69  000A  0004  185*        move.w  STalsrv(a1),4(sp)       ;Get alternate server from slot table
003E  4E75              186*        rts                             ;Return
                        187*
```

```
                        289* ;
                        290* ; OSSLTTYPE - Get device type for slot function
                        291* ;
                        292* ; FUNCTION OSsltType (slot: integer): slottype;
                        293* ;
0040                    294* OSsltType
0040  205F              295*         move.l  (sp)+,a0                ;Save return address
0042  301F              296*         move.w  (sp)+,d0                ;Get slot number
0044  548F              297*         addq.l  #2,sp                   ;Remove function result from stack
0046  5340              298*         subq.w  #1,d0                   ;Compute offset into slot table
0048  6D1C              299*         blt.s   slttyp8                 ;Error return if slot not valid
004A  0C40  0005        300*         cmpi.w  #5,d0                   ;*
004E  6C16              301*         bge.s   slttyp8                 ;Error return if slot not valid
0050  C0FC  0004        302*         mulu    #STinfoL,d0             ;*
0054  0640  000C        303*         addi.w  #STinfo,d0              ;*
0058  2278  0180        304*         move.l  pSysCom.w,a1            ;Get pointer to SysCom
005C  2269  003C        305*         move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0060  1F31  0001        306*         move.b  STtype(a1,d0.w),-(sp)   ;Get slot type for slot
0064  6002              307*         bra.s   slttyp9                 ;Return
                        308*                                         ;
0066  4227              309* slttyp8 clr.b   -(sp)                   ;Set slot type to no device
                        310*                                         ;
0068  4ED0              311* slttyp9 jmp     (a0)                    ;Return
                        312*
                        313* ;
                        314* ; OSEXTCRT - Check for external CRT function
                        315* ;
                        316* ; FUNCTION OSextCRT: boolean;
                        317* ;
006A                    318* OSextCRT
006A  205F              319*         move.l  (sp)+,a0                ;Save return address
006C  548F              320*         addq.l  #2,sp                   ;Remove function result from stack
006E  2278  0180        321*         move.l  pSysCom.w,a1            ;Get pointer to SysCom
0072  2269  0014        322*         move.l  SCdevtab(a1),a1         ;Get pointer to device table
0076  3019              323*         move.w  (a1)+,d0                ;Get number of devices
0078  2449              324*         move.l  a1,a2                   ;Compute last device pointer
007A  C0FC  0020        325*         mulu    #UTlen,d0               ;*
007E  D5C0              326*         adda.l  d0,a2                   ;*
0080  2269  0002        327*         move.l  UTiodrv(a1),a1          ;Get driver pointers
0084  246A  0002        328*         move.l  UTiodrv(a2),a2          ;*
0088  7001              329*         moveq   #1,d0                   ;Assume TRUE
008A  B5C9              330*         cmpa.l  a1,a2                   ;Driver [0] = driver [MAXDEV]?
008C  6700  0004        331*         beq     exrtn                   ;Yes, return
0090  7000              332*         moveq   #0,d0                   ;Set FALSE
0092  1F00              333* exrtn   move.b  d0,-(sp)                ;Set function result
0094  4ED0              334*         jmp     (a0)                    ;Return
                        335*
```

```
                 ;
                 ; 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                    391* OSomniDv
00CC  4267              392*        clr.w   -(sp)                ;Get number of devices
00CE  61C6              393*        bsr.s   OSmaxDev            ;*
00D0  301F              394*        move.w  (sp)+,d0            ;*
00D2  5740              395*        subq    #3,d0               ;Get device number
00D4  3F40  0004        396*        move.w  d0,4(sp)            ;Set function result
00D8  4E75              397*        rts                         ;Return
                        398*
                        399* ;
                        400* ; OSdcm2Dv - Get DTACOM2 driver device number function
                        401* ;
                        402* ; FUNCTION OSdcm2Dv: integer;
                        403* ,
00DA                    404* OSdcm2Dv
00DA  4267              405*        clr.w   -(sp)                ;Get number of devices
00DC  61B8              406*        bsr.s   OSmaxDev            ;*
00DE  301F              407*        move.w  (sp)+,d0            ;*
00E0  5940              408*        subq    #4,d0               ;Get device number
00E2  3F40  0004        409*        move.w  d0,4(sp)            ;Set function result
00E6  4E75              410*        rts                         ;Return
                        411*
                        412* ;
                        413* ; OSdcm1Dv - Get DTACOM1 driver device number function
                        414* ;
                        415* ; FUNCTION OSdcm1Dv: integer;
                        416* ,
00E8                    417* OSdcm1Dv
00E8  4267              418*        clr.w   -(sp)                ;Get number of devices
00EA  61AA              419*        bsr.s   OSmaxDev            ;*
00EC  301F              420*        move.w  (sp)+,d0            ;*
00EE  5B40              421*        subq    #5,d0               ;Get device number
00F0  3F40  0004        422*        move.w  d0,4(sp)            ;Set function result
00F4  4E75              423*        rts                         ;Return
                        424*
                        425* ,
                        426* ; OSsltDv - Get SLOTIO driver device number function
                        427* ;
                        428* ; FUNCTION OSsltDv: integer;
                        429* ;
00F6                    430* OSsltDv
00F6  4267              431*        clr.w   -(sp)                ;Get number of devices
00F8  619C              432*        bsr.s   OSmaxDev            ;*
00FA  301F              433*        move.w  (sp)+,d0            ;*
00FC  5D40              434*        subq    #6,d0               ;Get device number
00FE  3F40  0004        435*        move.w  d0,4(sp)            ;Set function result
0102  4E75              436*        rts                         ;Return
                        437*
```

```
                        439* ;
                        440* ; pOSuserID - Get Constellation user ID pointer
                        441* ;
                        442* ; FUNCTION pOSuserID: pointer;
                        443* ;
0104                    444* pOSuserID
0104  2F78  0184  0004  445*         move.l  pSysCom.w,4(sp)         ;Get pointer to SysCom
010A  06AF  0000  004C  446*         addi.l  #SCuserID,4(sp)         ;Get pointer to user ID
0110  0004
0112  4E75              447*         rts                             ;Return
                        448*
                        449* ;
                        450* ; pOScurWnd - Get current window record pointer
                        451* ;
                        452* ; FUNCTION pOScurWnd: pointer;
                        453* ;
0114                    454* pOScurWnd
0114  2078  0184        455*         move.l  pSysCom.w,a0            ;Get pointer to SysCom
0118  2F68  0044  0004  456*         move.l  SCcurrw(a0),4(sp)       ;Get current window pointer
011E  4E75              457*         rts                             ;Return
                        458*
                        459* ;
                        460* ; pOSsysWnd - Get system window record pointer
                        461* ;
                        462* ; FUNCTION pOSsysWnd (wndnbr: integer): pointer;
                        463* ;
0120                    464* pOSsysWnd
0120  205F              465*         move.l  (sp)+,a0                ;Save return address
0122  301F              466*         move.w  (sp)+,d0                ;Get system window number
0124  2F08              467*         move.l  a0,-(sp)                ;Restore return address
0126  E548              468*         lsl.w   #2,d0                   ;Get index to window pointer
0128  2078  0184        469*         move.l  pSysCom.w,a0            ;Get pointer to SysCom
012C  2068  0056        470*         move.l  SCwndtbl(a0),a0         ;Get pointer to window table
0130  2F70  0000  0004  471*         move.l  0(a0,d0),4(sp)          ;Get window pointer
0136  4E75              472*         rts                             ;Return
                        473*
                        474* ;
                        475* ; pOSdevNam - Get device name pointer
                        476* ;
                        477* ; FUNCTION pOSdevNam (unitnbr: integer): pointer;
                        478* ;
0138                    479* pOSdevNam
0138  205F              480*         move.l  (sp)+,a0                ;Save return address
013A  301F              481*         move.w  (sp)+,d0                ;Get unit number
013C  C0FC  0020        482*         mulu    #UTlen,d0               ;Compute entry index
0140  2F08              483*         move.l  a0,-(sp)                ;Restore return address
0142  2078  0184        484*         move.l  pSysCom.w,a0            ;Get pointer to SysCom
0146  2068  0014        485*         move.l  SCdevtab(a0),a0         ;Get pointer to device table
014A  D1FC  0000  0002  486*         adda.l  #2,a0                   ;Get pointer to device ID
0150  D1C0              487*         adda.l  d0,a0                   ;*
0152  D1FC  0000  0008  488*         adda.l  #UTdid,a0               ;*
0158  2F48  0004        489*         move.l  a0,4(sp)                ;Set function result
015C  4E75              490*         rts                             ;Return
                        491*
```

```
                        492* ,
                        493* , pOSdate - Get system date pointer
                        494* ,
                        495* , FUNCTION pOSdate. pointer,
                        496* ;
015E                    497* pOSdate
015E  2F78  0180  0004  498*         move.l  pSysCom.w,4(sp)         ;Get pointer to SysCom .
0164  06AF  0000  0020  499*         addi.l  #SCtoday,4(sp)          ;Get pointer to system date
016A  0004
016C  4E75              500*         rts                             ;Return
                        501*
```

```
                        503* ;
                        504* ; JSVECT - Jump to routine in system vector
                        505* ;
                        506* ; Parameters: D0.W - offset in system vector
                        507* ;
016E  2078  0180        508* JSVECT  MOVE.L  pSysCom.W,A0    , (A0) = syscom
0172  2068  0008        509*        MOVE.L  SCjtable(A0),A0 ; (A0) = sysvect
0176  2070  0000        510*        MOVE.L  0(A0,D0.W),A0   , (A0) = desired routine
017A  4ED0              511*        JMP     (A0)            , Go to it!
                        512*
                        513* ;
                        514* ; JUVECT - Jump to routine in user vector
                        515* ;
                        516* ; Parameters: D0.W - offset in user vector
                        517* ;
017C  2078  0180        518* JUVECT  MOVE.L  pSysCom.W,A0    , (A0) = syscom
0180  2068  001C        519*        MOVE.L  SCutable(A0),A0 , (A0) = uservect
0184  2070  0000        520*        MOVE.L  0(A0,D0.W),A0   ; (A0) = desired routine
0188  4ED0              521*        JMP     (A0)            , Go to it!
                        522*
                        523* ;
                        524* ; XGETDIR - Read a directory
                        525* ;
                        526* ; procedure xgetdir (fvid: vid; var fdir: directory, var DevBlocked. Boolean,
                        527* ;                    var fdevno. integer; var DevValid. Boolean), external;
                        528* ;
018A  7048              529* XGETDIR MOVEQ   #SVgetdir,D0
018C  60E0              530*        BRA.S   JSVECT
                        531*
                        532* ;
                        533* ; XPUTDIR - Write a directory
                        534* ;
                        535* ; procedure xputdir (var fdir. directory; fdevno: integer);
                        536* ;
018E  303C  0094        537* XPUTDIR MOVE.W  #SVputdir,D0
0192  60DA              538*        BRA.S   JSVECT
                        539*
```

```
                    541*        END

ACTIVE   00000006  IOEWNDFN 00000020  SCBOOTDV 00000036  STNMBR   00000000  UTDID    00000006
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
CSHPCH   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
IOENODSP 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   00004A+  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    0000F4+  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 00000184  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   000120+  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.

```
  1* ;
  2* ; file : timer.drv.text
  3* ; date : 20-SEPTEMBER-1982  kb
  4* ;
  5* ; INCLUDE FILES USED :
  6* ;       timer.clk.text                          ;HAS CALANDER CLOCK CODE
  7* ;       /ccos/os.gbl.asm.text                   ;OS GLOBAL EQUATES
  8* ;
  9* ; 04-06-82  kb  Added version date before TIMERDRV - entry point
 10* ; 04-23-82  kb  Changed IORESULT definitions to use the global file definitions
 11* ; 04-23-82  kb  Removed volume name from timer.clk.text include
 12* ; 06-07-82  kb  Changed for new rev. 4 processor board changes, will find the
 13* ;                  correct address to use (either $30FE1 or $30F81)
 14* ;                  added storage loc to save correct address
 15* ; 07-07-82  kb  Added Header to driver
 16* ; 07-07-82  kb  Fixed error in equates for different rev board address equates
 17* ; 09-20-82  lf  Changed write clock to stop/start clock in order to zero
 18* ;                  seconds and tenths fields (in TIMER.CLK.TEXT)
 19* ;
 20* ;
 21* ; INCLUDE OS GLOBALS HERE
 22* ;
253*            LIST       ON
254* ;
```

254* ;

```
                    258* ; EQUATES FOR ALL TIMER DRIVER SOFTWARE
                    259* ;
                    260* ; BIT NUMBER DEFINITIONS
                    261* ;
00000000            262* BITD0       EQU       0                   ;BIT 0
00000001            263* BITD1       EQU       1                   ;BIT 1
00000002            264* BITD2       EQU       2                   ;BIT 2
00000003            265* BITD3       EQU       3                   ;BIT 3
00000004            266* BITD4       EQU       4                   ;BIT 4
00000005            267* BITD5       EQU       5                   ;BIT 5
00000006            268* BITD6       EQU       6                   ;BIT 6
00000007            269* BITD7       EQU       7                   ;BIT 7
                    270* ;
                    271* ; TIMER INTERRUPT VECTOR ADDRESS
                    272* ,
00000074            273* VECTOR      EQU       $000074             ;INTERRUPT VECTOR #5
                    274* ;
                    275* ; TIMER TABLE INDICES
                    276* ;
00000000            277* TFLAGS      EQU       0                   ;TIMER TABLE FLAGS
00000002            278* PTRUSRTN    EQU       2                   ;POINTER TO USER SERVICE ROUTINE
00000006            279* TCOUNT      EQU       6                   ;# OF 50 MS. TICKS BEFORE CALL
00000008            280* TDWNCNT     EQU       8                   ;WORKING DOWN COUNTER
0000000A            281* REGA4       EQU       TDWNCNT+2           ;REGISTER A4 SAVE AREA
0000000E            282* REGA5       EQU       REGA4+4             ;REGISTER A5 SAVE AREA
                    283* ;
                    284* ; TIMER TABLE FLAGS BIT DEFINITIONS
                    285* ;
00000000            286* VALIDENT    EQU       BITD0               ;VALID ENTRY FLAG
00000001            287* CONT1SHT    EQU       BITD1               ;CONTINUOUS/1-SHOT MODE FLAG
00000002            288* SKIP1ST     EQU       BITD2               ;SKIP FIRST CALL FLAG
00000003            289* ENBLDSBL    EQU       BITD3               ;ENABLE/DISABLE FLAG
                    290* ;
                    291* ; BELL PARAMETER BLOCK INDICES
                    292* ;
00000000            293* FREQ        EQU       0                   ;FREQUENCY OF BELL
00000002            294* PATTERN     EQU       2                   ;PATTERN OF SPEAKER ON AND OFFS
00000004            295* DURATN      EQU       4                   ;DURATION OF BELL
                    296* ,
                    297* ; INTERNAL FLAG BIT DEFINITIONS
                    298* ;
00000000            299* SHUTOFF     EQU       BITD0               ;SHUTOFF BELL FLAG
                    300* ;
                    301* ; VIA ADDRESSES
                    302* ;
00030F77            303* ACR         EQU       $30F77              ;AUXILLARY CONTROL REGISTER
00030F7D            304* IER         EQU       $30F7D              ;INTERRUPT ENABLE REGISTER
00030F7B            305* IFR         EQU       $30F7B              ;INTERRUPT FLAGS REGISTER
00030F6D            306* T1LL        EQU       $30F6D              ;TIMER 1 LATCH LOW
00030F6F            307* T1LH        EQU       $30F6F              ;TIMER 1 LATCH HIGH
00030F69            308* T1CL        EQU       $30F69              ;TIMER 1 COUNTER LOW - READ ONLY
00030F6B            309* T1CH        EQU       $30F6B              ;TIMER 1 COUNTER HIGH
00030F71            310* T2LL        EQU       $30F71              ;TIMER 2 LATCH LOW
00030F73            311* T2CH        EQU       $30F73              ;TIMER 2 COUNTER HIGH
```

```
00030F75          312* SHIFTREG   EQU     $30F75          ;SHIFT REGISTER
                  313* ;
                  314* ; VIA REGISTER VALUES
                  315* ,
00000040          316* ACRBYTE    EQU     $40             ;ACR DATA - T1 FREE RUN DISABLE PB7
00000010          317* RUNT2     EQU     $10             ;MASK TO COUNT DOWN T2
000000EF          318* STOPT2    EQU     $EF             ;COMPLEMENTED RUNT2 TO STOP T2
0000007F          319* DISABL    EQU     $7F             ;DISABLE ALL INTERRUPTS
000000C0          320* ENBLT1    EQU     $C0             ;ENABLE IRQ FOR T1
000000FF          321* CLEAR     EQU     $FF             ;CLEAR ALL IFR STAT BITS
00000020          322* T2INT     EQU     $20             ;TIMER #2 INTERRUPT FLAG BIT
0000C350          323* TIME      EQU     50000           ;50,000 MICRO SECONDS
000000C3          324* TIMEH     EQU     TIME/256        ;HI ORDER BYTE OF TIME VALUE
00000050          325* TIMEL     EQU     TIME-(TIMEH*256)  ,LOW ORDER BYTE OF TIME VALUE
                  326* ,
                  327* ; CONTEXT SWITCHING DEFINITIONS
                  328* ;
0000005C          329* SPNDFLG   EQU     SCsusreq        ;SUSPEND FLAG
0000005A          330* SPWAITC   EQU     SCsusinh        ,WAIT SUSPEND COUNTED SEMAPHORE
00000002          331* CURPROC   EQU     SCprocno        ;CURRENT PROCESS # INDEX
000000BB          332* PFTBL     EQU     $BB             ;PTR TO PROCESS TABLE
000000BB          333* SCHEDPTR  EQU     $BB             ;PTR TO ENTRY OF SCHEDULER
000000BB          334* SCHDA4    EQU     $BB             ;REG. A4 VALUE FOR SCHEDULER
000000BB          335* SCHDA5    EQU     $BB             ;REG. A5 VALUE FOR SCHEDULER
                  336* ,
000000CC          337* PTLEN     EQU     $CC             ;LENGTH OF PROCESS TABLE ENTRY
0000000F          338* NUMREGS   EQU     15              ;NUMBER OF REGISTERS SSAVED IN PTBL
0000003C          339* PTPC      EQU     NUMREGS*4       ;PROCESS TABLE-PC FIELD
00000040          340* PTSR      EQU     PTPC+4          ;PROCESS TABLE-SR FIELD
                  341* ;
00002700          342* SCHEDSR    EQU     $2700           ;SCHEDULER SR-NO INTERRUPTS
                  343* ;
                  344* ; IORESULT ERROR CODES
                  345* ;
00000036          346* INVPRM     EQU      IOEmiopm       ;INVALID UNIT I/O PARAMETER
00000003          347* NOTLEGIT   EQU      IOEioreq       ;NOT LEGITIMATE CALL
00000032          348* INVTBLID   EQU      IOEtblid       ;INVALID TABLE ENTRY ID
00000033          349* TBLFULL    EQU      IOEtblfl       ;TIMER TABLE FULL
00000038          350* INVFNC     EQU      IOEfnccd       ;invalid function code
                  351* ;
                  352* ; MISCELLANEOUS EQUATES
                  353* ;
00000006          354* UNMCMD     EQU     6               ;UNMOUNT COMMAND CODE
00000004          355* ENABLEC    EQU     4               ;ENABLE FUNCTION CODE
00000001          356* CARRYST    EQU     $01             ;CARRY SET IN CCR
00000001          357* ON         EQU     1
00000000          358* OFF        EQU     0
```

```
360* ; TIMER INTERRUPT SERVICE ROUTINE
361* , INTERNAL REGISTER USEAGE :
362* ;         A0 - TEMP
363* ;         A1 - TEMP
364* ;         A2 - TIMER TABLE ADDRESS
365* ,         A3 - ADDRESS OF CURRENT ENTRY'S FLAG'S LOW ORDER BYTE
366* ;
367* ;         D0 - TEMP
368* ;         D1 - TEMP
369* ;         D2 - INDEX TO CURRENT ENTRY IN TIMER TABLE
370* ;
```

```
                        372* ;
                        373* , TIMER INTERRUPT SERVICE ROUTINE
                        374* ; THIS ROUTINE IS INVOKED WHEN THE 50 MILLISECOND INTERVAL TIMER INTERRUPT
                        375* ; OCCURS.  IT CHECKS EACH ENTRY OF THE TIMER TABLE TO SEE IF IT'S USER SERVICE
                        376* ; ROUTINE SHOULD BE CALLED.
                        377* ;
0000                    378* TIMINT
0000  48E7  FFFE        379*            MOVEM.L     D0-D7/A0-A6,-(SP)       ;SAVE USER'S REGISTERS
0004  1039  0003  0F69  380*            MOVE.B      T1CL.L,D0               ;RESET VIA IFR T2 BIT
                        381* ;
                        382* , for i := 1 to NUMENTS do
                        383* ,
000A  45FA  0524+       384*            LEA         TIMERTBL,A2             ;ADDRESS OF TIMER TABLE
000E  4282              385*            CLR.L       D2                      ;START WITH FIRST ENTRY
                        386* ;
                        387* ; IF HAVE VALID ENTRY THAT IS NOT DISABLE THEN SEE IF SHOULD CALL USER SERVICE ROUTINE
                        388* ;
0010  47F2  2001        389* TINEXT      LEA         TFLAGS+1(A2,D2.W),A3    ;ADDRESS OF CURRENT FLAGS + 1
0014  0813  0000        390*            BTST        #VALIDENT,(A3)          ;VALID ENTRY?
0018  6734              391*            BEQ.S       TICHKNIT                ;NO, SEE IF ANOTHER ENTRY
001A  0813  0003        392*            BTST        #ENBLDSBL,(A3)          ;IS ENTRY ENABLED?
001E  662E              393*            BNE.S       TICHKNIT                ;NO, SEE IF ANOTHER ENTRY
                        394* ;
                        395* ; GOT VALID ENTRY - TEST IF SHOULD CALL USER SERVICE ROUTINE
                        396* ;
0020  0472  0001  2008  397*            SUBI.W      #1,TDWNCNT(A2,D2.W)     ;DOWN COUNT
0026  6626              398*            BNE.S       TICHKNIT                ;NOT DONE, SEE IF ANOTHER ENTRY
                        399* ;
0028  48E7  2030        400*            MOVEM.L     D2/A2-A3,-(SP)          ;SAVE WORKING REGISTERS
002C  2872  200A        401*            MOVEA.L     REGA4(A2,D2.W),A4       ;SETUP USERS A4 AND A5
0030  2A72  200E        402*            MOVEA.L     REGA5(A2,D2.W),A5       ;REGISTERS
0034  2072  2002        403*            MOVEA.L     PTRUSRTN(A2,D2.W),A0    ;ADDRESS USER SERVICE ROUTINE
0038  4E90              404*            JSR         (A0)                    ;CALL USER SERVICE ROUTINE
003A  4CDF  0C04        405*            MOVEM.L     (SP)+,D2/A2-A3          ;RESTORE REGISTERS
                        406* ;
                        407* ; RESET DOWN COUNTER - ASSUME CONTINUOUS MODE
                        408* ;
003E  35B2  2006  2008  409*            MOVE.W      TCOUNT(A2,D2.W),TDWNCNT(A2,D2.W)
                        410* ;
                        411* ; IF ENTRY IS IN 1 SHOT MODE THEN DELETE THE ENTRY.
                        412* ;
0044  0813  0001        413*            BTST        #CONTISHT,(A3)          ;1 SHOT MODE?
0048  6704              414*            BEQ.S       TICHKNIT                ;NO, SEE IF ANOTHER ENTRY
004A  0893  0000        415*            BCLR        #VALIDENT,(A3)          ;YES, DELETE ENTRY
                        416* ;
                        417* ; INCREMENT INDEX - IF NOT PAST END OF TABLE THEN DO NEXT ENTRY
                        418* ;
004E  0642  0012        419* TICHKNIT    ADDI.W      #TIMTLEN,D2             ;INDEX TO NEXT ENTRY
0052  0C42  00B4        420*            CMPI.W      #TABLELN,D2             ;PAST END OF TABLE?
0056  66B8              421*            BNE.S       TINEXT                  ;NO, DO NEXT ENTRY
                        422* ;
                        423* ; SEE IF SHOULD DO CONTEXT SWITCH
                        424* ;
                        425* ;           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
```

```
                        431* ;
                        432* ; DOCS - DO CONTEXT SWITCH
                        433* ;          ENTRY . MUST BE A GOTO CALL VIA A JUMP OR BRA NOT A SUBROUTINE
                        434* ;                  CALL. NO EXTRA STUFF ON STACK.
                        435* ,                  THE TOP OF STACK MUST BE THE USER'S REGISTERS
                        436* ,                  (A0) = POINTER TO SYSCOM
                        437* ;
005E  4228  005C        438* DOCS       CLR.B     SPNDFLG(A0)             ;CLEAR SUSPEND FLAG
                        439* ;
                        440* , SAVE EXISTING PROCESSES CONTEXT (PARTIAL, SCHEDULER DOES REST)
                        441* ;
0062  3028  0002        442*            MOVE.W    CURPROC(A0),D0          ;GET CURRENT PROCESS #
0066  C0FC  00CC        443*            MULU      #PTLEN,D0               ;CALC INDEX TO PROCESS TABLE ENTRY
006A  2268  00BB        444*            MOVEA.L   PPTBL(A0),A1            ;ADDRESS OF PROCESS TABLE
006E  45F1  0000        445*            LEA       0(A1,D0.W),A2           ;ADDRESS OF ENTRY
0072  720E              446*            MOVEQ     #NUMREGS-1,D1           ;COUNT OF POPS
                        447* ;
0074  24DF              448* DCSMOVR    MOVE.L    (SP)+,(A2)+             ;SAVE REGISTERS IN ENTRY
0076  51C9  FFFC        449*            DBF       D1,DCSMOVR              ;IN ORDER D0-A6
                        450* ;
007A  355F  0040        451*            MOVE.W    (SP)+,PTSR(A2)          ;SAVE SR AND PC OF CURRENT
007E  355F  003C        452*            MOVE.W    (SP)+,PTPC(A2)          ;PROCESS
                        453* ;
                        454* , CALL SCHEDULER VIA A FAKED RTE
                        455* ;
0082  2F28  00BB        456*            MOVE.L    SCHEDPTR(A0),-(SP)      ;ENTRY POINT TO SCHEDULER
0086  3F3C  1700        457*            MOVE.W    #SCHEDSR,-(SP)          ;SR FOR SCHEDULER
                        458* ;
008A  2868  00BB        459*            MOVE.L    SCHDA4(A0),A4           ,SCHEDULER IS A PASCAL GLOBAL SUBROUTINE
008E  2A68  00BB        460*            MOVE.L    SCHDA5(A0),A5           ;NEEDS ITS VALUES FOR A4 & A5
0092  4E73              461*            RTE
```

```
                        463* ;
                        464* ; CHKCS - SEE IF SHOULD DO A CONTEXT SWITCH
                        465* ;           EXIT : (NE) - DON'T DO CONTEXT SWITCH
                        466* ;                  (EQ) - DO SWITCH
                        467* ;                  (A0) = POINTER TO SYSCOM
                        468* ; IF (SUSPEND FLAG IS CLEAR) THEN DON'T DO SWITCH
                        469* ;
0094  2078  0180        470* CHKCS       MOVE.L     PSYSCOM.W,A0            ;ADDRESS OF SYSCOM
0098  4A28  005C        471*            TST.B      SPNDFLG(A0)             ;FLAG CLEAR?
009C  6706              472*            BEQ.S      CCSDONT                 ;YES
                        473* ;
                        474* ; IF (SUSPEND WAIT COUNTED SEMAPHORE = 0) THEN DO CONTEXT SWITCH
                        475* ,
009E  4A28  005A        476*            TST.B      SPWAITC(A0)
00A2  6002              477*            BRA.S      CCSEXIT
                        478* ;
00A4  7001              479* CCSDONT     MOVEQ      #1,D0                   ;FORCE DONT (NE)
00A6  4E75              480* CCSEXIT     RTS
```

```
482* ;
483* ; UNIT I/O PARAMETER PASSING DEFINITION
484* ;
485* ;   COMMAND       UNIT   ADDR   COUNT   BLOCK   MODE    IORESULT   BUSY
486* , 0 - INSTALL     D0.W                                   D7.W
487* , 1 - READ        D0.W   D1.L   D2.W             D5.W    D7.W
488* , 2 - WRITE       D0.W   D1.L   D2.W             D5.W    D7.W
489* , 3 - CLEAR       D0.W                                   D7.W
490* , 4 - BUSY        D0.W                                   D7.W       D6.W
491* , 5 - STATUS      D0.W   D1.L   D2.W                     D7.W
492* , 6 - UNMOUNT     D0.W                                   D7.W
493* ,
494* ; ALL REGISTER VALUES ON ENTRY ARE SAVED AND RESTORED EXCEPT D6 & D7.
495* , INTERNAL REGISTER USEAGE :
496* ;           A0 - TEMP  (GLOBAL)
497* ,           A1 - TEMP  (GLOBAL)
498* ;           A2 - ADDRESS OF TIMER TABLE (GLOBAL)
499* ;           A3 - ADDRESS OF USER'S BUFFER ADDRESS (GLOBAL)
500* ,           A4 - ADDRESS OF INTERNAL FLAGS BYTE (BELL)
501* ,           A5 - ADDRESS OF VIA SHIFT REGISTER (BELL)
502* ;           A6 - ADDRESS OF VIA INTERRUPT ENABLE REGISTER (INSTALL)
503* ;
504* ;           D0 - TEMP
505* ;           D1 - TEMP
506* ,           D2 - COUNT OR CONTROL
507* ;           D3 - TABLE ENTRY INDEX
508* ,           D4 - PATTERN FOR BELL
509* ,
```

```
                        511* ;
                        512*            GLOBAL      TIMERDRV
                        513* ,
                        514* ; TIMER DRIVER
                        515* ;
00A8                    516* TIMERDRV
00A8  6014              517*            BRA.S       TIMR001                 ,*070782* JUMP AROUND HEADER
00AA  00                518*            DATA.B      0                       ;DEVICE NOT BLOCKED
00AB  1F                519*            DATA.B      31                      ;VALID CMDS - NOT UNITSTATUS
00AC  52 0A 16 00       520*            DATA.B      82,10,22,0              ;DATE
00B0  0C                521*            DATA.B      hmlen                   ,HEADER MSG LENGTH
00B1  54494D4552206472  522* xxx010      DATA.B      'TIMER driver'          ;HEADER MSG
00B9  69766572
      0000000C          523* hmlen       EQU         *-xxx010
                        524* ;
00BE  0C44  0006        525* TIMR001     CMPI.W      #UNMCMD,D4              ;VALID COMMAND
00C2  621C              526*            BHI.S       TIMDERR                 ;NO
                        527* ;************ wait till change to D6 for busy return ************
                        528* ;****       MOVEM.L     D0-D5/A0-A6,-(SP)       ;SAVE REGISTERS
                        529* ;*************************************************************
00C4  48E7  7EFE        530*            movem.l     D1-D6/A0-A6,-(SP)       ,*** temp* for busy return in D0
00C8  4287              531*            CLR.L       D7                      ;CLEAR IORESULT
00CA  2641              532*            MOVEA.L     D1,A3                   ;ADDRESS OF USERS BUFFER
00CC  43FA  0018+       533*            LEA         TIMDTBL,A1              ;TURN THE COMMAND INTO A
00D0  E34C              534*            LSL.W       #1,D4                   ;INDEX TO THE FUNCTION
00D2  3831  4000        535*            MOVE.W      0(A1,D4.W),D4
00D6  4EB1  4000        536*            JSR         0(A1,D4.W)              ;DO FUNCTION
                        537* ;************ wait till change to D6 for busy return ************
                        538* ;****       MOVEM.L     (SP)+,D0-D5/A0-A6       ;RESTORE SAVED REGISTER VALUES
                        539* ;*************************************************************
00DA  4CDF  7F7E        540*            movem.l     (SP)+,D1-D6/A0-A6       ;*** temp* for busy return in D0
00DE  4E75              541*            RTS
                        542* ;
                        543* ; Invalid Command Error
                        544* ;
00E0  3E3C  0003        545* TIMDERR     MOVE.W      #NOTLEGIT,D7
00E4  4E75              546*            RTS
                        547* ;
                        548* ; THE TIMER DRIVER JUMP TABLE
                        549* ;
00E6  027E              550* TIMDTBL     DATA.W      TIMINST-TIMDTBL         ;UNITINISTALL
00E8  001A              551*            DATA.W      TIMRD-TIMDTBL           ;UNITREAD
00EA  01D0              552*            DATA.W      TIMWR-TIMDTBL           ;UNITWRITE
00EC  0014              553*            DATA.W      TIMCLR-TIMDTBL          ;UNITCLEAR
00EE  000E              554*            DATA.W      TIMBSY-TIMDTBL          ,UNITBUSY
00F0  02E6              555*            DATA.W      TIMST-TIMDTBL           ;UNITSTATUS
00F2  02D4              556*            DATA.W      TIMUNMT-TIMDTBL         ;UNITUNMOUNT
```

```
                        558* ,
                        559* ; TIMBSY - UNITBUSY
                        560* , BUSY FROM THE TIMER IS CURRENTLY UNDEFINED.
                        561* ,
00F4  3E3C  0003        562* TIMBSY      MOVE.W     #NOTLEGIT,D7
00F8  4E75              563*            RTS
                        564* ,
                        565* ; TIMCLR - UNITCLEAR
                        566* ; CLEAR THE TIMER IS CURRENTLY UNDEFINED.
                        567* ;
00FA  3E3C  0003        568* TIMCLR      MOVE.W     #NOTLEGIT,D7
00FE  4E75              569*            RTS
```

```
571* ;
572* ; TIMWR - UNITWRITE
573* ; TIMRD - UNITREAD
574* ;
575* ; CODE FOR CALANDER CLOCK : IN INCLUDE FILE timer.clk.text
576* ;
577*            INCLUDE    'TIMER.CLK.TEXT'
578* ;
579* ; file : timer.clk.text
580* ; date : 22-OCTOBER-1982  kb
581* ;
582* ; FILE IS AN INCLUDE FILE FOR TIMER.DRV.TEXT , THE TIMER DRIVER.
583* ; THIS IS THE UNITREAD AND UNITWRITE CODE FOR THE CALANDER CLOCK IN THE
584* ; TIMER DRIVER.
585* ;
586* ; 04-23-82  kb  Changed IORESULT definitions to use the global file definitions
587* ; 06-07-82  kb  Changed for new rev. 4 processor board changes, will find the
588* ;                  correct address to use (either $30FE1 or $30F81)
589* ; 07-07-82  kb  Changed error in RV3 and RV4 address equates, were reversed
590* ; 09-20-82  lf  Changed write clock to stop/start clock in order to zero
591* ;                  seconds and tenths fields
592* ; 10-22-82  kb  Changed READCR so it reads 4 times with a MOVEP.L instruction.
593* ;                  Checks to make sure the low order 3 bytes, all except the
594* ;                  first read are the same.  If they are not it rereads the
595* ;                  register.
596* ;
```

```
                    598* ;
                    599* ; EQUATES FOR THE CLOCK ROUTINES
                    600* , PARAMTER BLOCK INDICES - RANGE FOR PARAMETER IN PARENTHESIS
                    601* ;
00000000            602* DAYOFWK     EQU     0                   ;DAY OF THE WEEK (1-7)
00000002            603* MONTH      EQU     DAYOFWK+2           ;MONTH (1-12)
00000004            604* DAY        EQU     MONTH+2             ;DAY (1-31)
00000006            605* HOUR       EQU     DAY+2               ;HOURS (0-23)
00000008            606* MINS       EQU     HOUR+2              ;MINUTES (0-59)
0000000A            607* SECS       EQU     MINS+2              ;SECONDS (0-59)
0000000C            608* TENTHS     EQU     SECS+2              ;TENTHS OF SECONDS (0-9)
0000000E            609* LEAPYR     EQU     TENTHS+2            ;LEAP YEAR (0-3)
                    610* ,
0000000E            611* LENPBR     EQU     TENTHS+2            ;LENGTH OF READ PARAMETER BLOCK
00000007            612* NUMRP      EQU     LENPBR/2            ;NUMBER OF READ PARAMETERS
                    613* ;
00000010            614* LENPBW     EQU     LEAPYR+2            ;LENGTH OF WRITE PARAMETER BLOCK
00000008            615* NUMWP      EQU     LENPBW/2            ;NUMBER OF WRITE PARAMETERS
                    616* ,
                    617* , CLOCK REGISTERS
                    618* ,
00030F81            619* RV4ADDR    EQU     $30F81              ;SELECT/ADDRESS LATCH   **new board address
00030FE1            620* RV3ADDR    EQU     $30FE1              ;SELECT/ADDRESS LATCH  **old board address
00030D01            621* RWREG      EQU     $30D01              ;READ/WRITE CLOCK REGISTERS
0000000F            622* INTREG     EQU     15                  ;CLOCK INTERRUPT REG ADDR
0000000E            623* STRTSTOP   EQU     14                  ;START/STOP REGISTER
0000000D            624* LYREG      EQU     13                  ;LEAP YEAR REGISTER
00000001            625* TENTHSC    EQU      1                  ;TENTH OF SECONDS REGISTER (change 6/7)
                    626* ;
0000000F            627* RDERR      EQU     $0F                 ;REGISTER VALUE WHEN READ WHEN UPDATE
00000010            628* DSELCT     EQU     $10                 ;DESELECT CHIP
                    629* ;
                    630* ; IORESULT CODES
                    631* ;
00000037            632* PBLENER    EQU     IOEprmln            ;PARAMETER BLOCK WRONG LENGTH
00000039            633* CLOCKERR   EQU     IOEclkmf            ;CLOCK NOT WORKING
                    634* ;
```

```
                        636* ;
                        637* ; TIMRD - UNITREAD OF CLOCK
                        638* ;
                        639* ; RETURN TO THE USER THE TIME IN THE REAL-TIME-CLOCK.
                        640* ; USER PASSES PARAMETER BLOCK POINTER IN D1 OF WHERE TO PUT THE TIME INFO.
                        641* ; TIME IS RETURNED IN BINARY, INTEGERS.   THE PARAMETER BLOCK HAS THE FORM
                        642* ;
                        643* ; type ReadClockParameter = record
                        644* ;                             DayofWeek : integer;
                        645* ;                             Month : integer;
                        646* ;                             Day : integer;
                        647* ;                             Hour : integer;
                        648* ;                             Mins : integer;
                        649* ;                             Secs : integer;
                        650* ;                             Tenths : integer;
                        651* ;                           end;
                        652* ;
0100                    653* TIMRD
0100  2C41               654*            MOVE.L     D1,A6                ;A6 = PARAMETER BLOCK PTR
0102  2802               655*            MOVE.L     D2,D4                ;SAVE USER LENGTH
0104  6138               656*            BSR.S      RDCLOCK              ;READ THE CLOCK
0106  4A47               657*            TST.W      D7
0108  662C               658*            BNE.S      TRDEXIT              ;CAN'T READ CLOCK
                        659* ;
                        660* ; CONVERT AND PUT RESULT IN PARAMETER BLOCK
                        661* ;
010A  0C44  000E        662*            CMPI.W     #LENPBR,D4           ;IS PARAMETER BLOCK LONG ENOUGH
010E  6528              663*            BCS.S      TRDERR               ;NO
0110  4BFA  03EC+       664*            LEA        NUMBER,A5            ;# OF NIBBLES IN PARAMETER
0114  49FA  03DC+       665*            LEA        DETAIL,A4            ;REGISTER ARRAY INDICES
0118  47FA  04DA+       666*            LEA        REGARRAY,A3          ;REGISTER ARRAY
011C  4283              667*            CLR.L      D3                   ;INDEX INTO PARAM BLOCK
011E  4284              668*            CLR.L      D4                   ;INDEX INTO NUMBER ARRAY
0120  4286              669*            CLR.L      D6                   ;INDEX INTO DETAIL ARRAY
                        670* ;
0122  1035  4000        671* TRDGETP     MOVE.B     0(A5,D4.W),D0        ;# OF NIBBLES PARAMETER
0126  6158              672*            BSR.S      CVTOUT               ;CONVERT - RETURNS IN D1
0128  3D81  3000        673*            MOVE.W     D1,0(A6,D3.W)        ;STORE PARAMETER
012C  5443              674*            ADDQ.W     #2,D3                ;NEXT PARAMETER
012E  5204              675*            ADDQ.B     #1,D4                ;DO 7 PARAMETERS
0130  0C44  0007        676*            CMPI.W     #NUMRP,D4
0134  66EC              677*            BNE.S      TRDGETP              ;DO AGAIN
0136  4E75              678* TRDEXIT     RTS
                        679* ;
                        680* ; ERROR - PARAMETER BLOCK THE WRONG LENGTH
                        681* ;
0138  3E3C  0037        682* TRDERR      MOVE.W     #PBLENER,D7
013C  4E75              683*            RTS
```

```
                        685* ,
                        686* , RDCLOCK - READ ALL THE CLOCK REGISTERS INTO THE REGISTER ARRAY
                        687* ,
013E  6130              688* RDCLOCK     BSR.S     LDADDR                  ,GET CHIP ADDRESSES IN A0&A1
0140  7C07              689*             MOVEQ     #7,D6                   ,COUNT OF FAILURES
                        690* ,
                        691* , READ REGISTERS INTO ARRAY
                        692* ,
0141  5306              693* RDCRST      SUBQ.B    #1,D6
0144  6B24              694*             BMI.S     RDCERR                  ,RETRIED TO MANY TIMES
0146  7001              695*             MOVEQ     #1,D0                   ;D0 = REGISTER #
0148  45FA  04AA+       696*             LEA       REGARRAY,A2
                        697* ,
014C  6166              698* RDCREG      BSR.S     READCR                  ,READ A SINGLE REGISTER REURNS IN D2
014E  14C2              699*             MOVE.B    D2,(A2)+                ,PUT IN ARRAY
0150  5200              700*             ADDQ.B    #1,D0
0152  0C00  000D        701*             CMPI.B    #RARDLEN+1,D0           ,IF DONE 12 TIMES STOP
0156  66F4              702*             BNE.S     RDCREG                  ,DO AGAIN
                        703* ,
                        704* , IF ANY REGISTER READ = $0F THEN READ REGISTERS WHEN TICKED AND MUST RESTART
                        705* ,
0158  700B              706*             MOVEQ     #RARDLEN-1,D0           ,DO 12 TIMES
015A  45FA  0498+       707*             LEA       REGARRAY,A2
                        708* ,
015E  0C1A  000F        709* RDDCHK      CMPI.B    #RDERR,(A2)+            ,BAD
0162  67DE              710*             BEQ.S     RDCRST                  ,YES, REREAD CLOCK
0164  51C8  FFF8        711*             DBF       D0,RDDCHK
0168  4E75              712*             RTS
                        713* ,
                        714* ; ERROR - CHIP NOT WORKING
                        715* ,
016A  3E3C  0039        716* RDCERR      MOVE.W    #CLOCKERR,D7
016E  4E75              717*             RTS
                        718* ,
                        719* ; LDADDR - GET LATCH ADDRESS IN A0 AND R/W CLOCK ADDRESS IN A1
                        720* ,
0170  207A  047E+       721* LDADDR     MOVEA.L   ADDRREG,A0              ; GET saved address (change 6/7)
0174  2008              722*             MOVE.L    A0, D0                  ; (change 6/7)
0176  67F2              723*             BEQ.S     RDCERR                  ;NO CLOCK CHIP - ERROR EXIT (change 6/7)
0178  43F9  0003  0D01  724*             LEA       RWREG.L,A1
017E  4E75              725*             RTS
```

```
                        727* ;
                        728* ; CVOUT - CONVERT REGISTERS TO 1 PARAMETER
                        729* ;          ENTRY : (D0) = # OF REGISTERS TO USE (# OF DETAIL ELEMENTS)
                        730* ;                  DETAIL(D6) = REGISTER TO USE TO MAKE PARAMETER
                        731* ;                  (A4) = ADDRESS OF THE INDICES OF REGARRAY FOR EACH PARAMETER
                        732* ;                  (A3) = ADDRESS OF THE REGISTER ARRAY
                        733* ;          EXIT  : (D1) = PARAMETER CONVERTED
                        734* ;                  (D6) = UPDATED TO NEXT DETAIL ELEMENT FOR NEXT PARAMETER
                        735* ;
0180  4282              736* CVTOUT     CLR.L     D2
0182  4281              737*           CLR.L     D1
0184  1434  6000        738*           MOVE.B    0(A4,D6.W),D2          ;GET REGARRAY INDEX
0188  5206              739*           ADDQ.B    #1,D6                  ;UPDATE INDEX
018A  1233  2000        740*           MOVE.B    0(A3,D2.W),D1          ;FIRST NIBBLE
                        741* ;
018E  5300              742*           SUBQ.B    #1,D0                  ;IF NUMBER OF REGS=1  THEN
0190  670E              743*           BEQ.S     CVOEXIT                ;THEN DONE-IS VALID BINARY
                        744* ;
0192  E909              745*           LSL.B     #4,D1                  ;MOVE TO HI NIBBLE
0194  1434  6000        746*           MOVE.B    0(A4,D6.W),D2          ;INDEX TO NEXT REGISTER
0198  5206              747*           ADDQ.B    #1,D6                  ;UPDATE INDEX
019A  8233  2000        748*           OR.B      0(A3,D2.W),D1          ;PUT IN LOW NIBBLE
                        749* ;
019E  6102              750*           BSR.S     CBCDBIN                ;CONVERT BCD TO BINARY
01A0  4E75              751* CVOEXIT    RTS
```

```
                        753* ,
                        754* ; CBCDBIN - CONVERT 1 BYTE OF BCD (2 DIGITS) TO 1 BYTE OF BINARY
                        755* ;          ENTRY : (D1) = BCD BYTE OF DIGITS
                        756* ;          EXIT  . (D1) = BINARY BYTE
                        757* ;
01A2  4282              758* CBCDBIN     CLR.L     D2
01A4  1401              759*             MOVE.B    D1,D2                    ,COPY OF BCD
01A6  0202  000F        760*             ANDI.B    #$0F,D2                  ;LOW ORDER DIGIT
01AA  E849              761*             LSR.W     #4,D1                    ,MOVE OVER HI DIGIT
01AC  C2FC  000A        762*             MULU      #10,D1                   ,MAKE 10*DIGIT
01B0  D242              763*             ADD.W     D2,D1                    ;MAKE FULL NUMBER
01B2  4E75              764*             RTS
```

```
                        766* ;
                        767* ; Changed routine on 10/22/82 kb
                        768* ; READCR - READ CLOCK REGISTER
                        769* ;       ENTRY : (A0) = LATCH ADDRESS
                        770* ;               (A1) = R/W CLOCK ADDRESS
                        771* ;               (D0) = REGISTER ADDRESS
                        772* ,       EXIT  : (D2) = REGISTER VALUE READ
                        773* ,
                        774* ;  The movep.l instruction works because the I/O address space for the
                        775* ;  clock is not fully decoded.
                        776* ;
01B4  48E7  0E00        777* READCR      MOVEM.L   D4-D6,-(SP)             ;save regs                  *kb 10/22/8
2*
                        778*
01B8  6124              779* RDCR10      BSR.S     SELREG                  ;DESELECT THEN SELECT ADDRESS  *kb 10/22/8
2*
01BA  0549  0000        780*            MOVEP.L   0(A1),D2                ;read reg 4 times           *kb 10/22/8
2*
01BE  10BC  0010        781*            MOVE.B    #DSELCT,(A0)            ;DESELECT CHIP
01C2  0282  0F0F 0F0F   782*            ANDI.L    #$0F0F0F0F,D2           ;clear hi nibbles of all bytes *kb 10/22/8
2*
                        783* ;                                                                       *kb 10/22/8
2*
                        784* ; make sure all bytes read are the same                                  *kb 10/22/8
2*
                        785* ;                                                                       *kb 10/22/8
2*
01C8  7801              786*            MOVEQ     #1, D4                  ;ignore hi order byte, 1st read *kb 10/22/8
2*
01CA  2A02              787*            MOVE.L    D2,D5                   ;save read value            *kb 10/22/8
2*
01CC  1C05              788*            MOVE.B    D5, D6                  ;compare all to last read   *kb 10/22/8
2*
01CE                    789* RDCR20                                       ;                           *kb 10/22/8
2*
01CE  E08D              790*            LSR.L     #8, D5                  ;chk next byte              *kb 10/22/8
2*
01D0  BC05              791*            CMP.B     D5, D6                  ;are they the same?         *kb 10/22/8
2*
01D2  66E4              792*            BNE.S     RDCR10                  ;No, read reg again         *kb 10/22/8
2*
01D4  51CC  FFF8        793*            DBF       D4, RDCR20              ;Do until checked all 3 bytes  *kb 10/22/8
2*
01D8  4CDF  0070        794*            MOVEM.L   (SP)+,D4-D6             ;restore regs               *kb 10/22/8
2*
01DC  4E75              795*            RTS
                        796* ;
                        797* , SELREG - DESELECT THEN SELECT CHIP REGISTER
                        798* ;           ENTRY : (D0) = CLOCK REGISTER ADDRESS
                        799* ,                   (A0) = LATCH ADDRESS
                        800* ,
01DE  7210              801* SELREG      MOVEQ     #DSELCT,D1
01E0  8200              802*            OR.B      D0,D1
01E2  1081              803*            MOVE.B    D1,(A0)                 ;DESELECT CHIP BY SETTING D4
01E4  1080              804*            MOVE.B    D0,(A0)                 ;SELECT ADDRESS
01E6  4E75              805*            RTS
```

```
                        807* ;
                        808* ; INITCLK - PROCEDURE CALLED BY UNITINSTALL CODE (TIMINST) TO INITIALIZE
                        809* ;           THE CLOCK CHIP.
                        810* ;   ASSUMES THAT THE FOLLOWING CODE DOES NOT RESET THE CLOCK.???????
                        811* ;
01E8                    812* INITCLK
                        813* ;
                        814* ; START CLOCK - MUST DO IT FOR BOTH ADDRESSES
                        815* ;
01E8  43F9 0003 0D01    816*            LEA       RWREG.L, A1           ;R/W ADDRESS          (change 6/7)
01EE  41F9 0003 0FE1    817*            LEA       RV3ADDR.L, A0         ;DO OLD ADDRESS FIRST (change 6/7)
01F4  612A              818*            BSR.S     STRTCLK               ;                     (change 6/7)
01F6  41F9 0003 0F81    819*            LEA       RV4ADDR.L, A0         ;DO NEW ADDRESS       (change 6/7)
01FC  6122              820*            BSR.S     STRTCLK               ;                     (change 6/7)
                        821* ;                                                                (change 6/7)
                        822* ; FIND CORRECT ADDRESS OF THIS MACHINES PROCESSOR BOARD           (change 6/7)
                        823* ;                                                                (change 6/7)
01FE  6146              824*            BSR.S     FINDADDR              ;                     (change 6/7)
0200  6618              825*            BNE.S     INITEXIT              ;ERROR - NO CHIP      (change 6/7)
                        826* ;
                        827* ; INITIALIZE CHIP
                        828* ;
0202  6100 FF6C         829*            BSR       LDADDR                ;GET CLOCK REGISTER ADDRESES
0206  4280              830*            CLR.L     D0                    ;REG ADDRESS
0208  4282              831*            CLR.L     D2                    ;DATA
020A  6100 0086         832*            BSR       WRITECR               ;PUT IN NON-TEST MODE
                        833* ;
                        834* ; CLEAR INTERRUPTS
                        835* ;
020E  700F              836*            MOVEQ     #INTREG,D0            ;ADDRESS
0210  4282              837*            CLR.L     D2                    ;DATA
0212  617E              838*            BSR.S     WRITECR
0214  619E              839*            BSR.S     READCR                ;READ 3 TIMES TO RESET
0216  619C              840*            BSR.S     READCR
0218  619A              841*            BSR.S     READCR
021A  4E75              842* INITEXIT    RTS                             ;(change 6/7)
```

```
                        844* ;
                        845* ; STOPCLK - STOP CLOCK PROCEDURE                                    (09-20-82)
                        846* ;       ASSUMES A0 AND A1 ARE INITIALIZED.                         (09-20-82)
                        847* ;                                                                  (09-20-82)
021C  7400              848* STOPCLK    MOVEQ     #0,D2                 ;DATA                  (09-20-82)
021E  6002              849*            BRA.S     ST10                  ;                      (09-20-82)
                        850* ;                                                                  (09-20-82)
                        851* ; STRTCLK - START CLOCK PROCEDURE                                  (06-07-82)
                        852* ;       ASSUMES A0 AND A1 ARE INITIALIZED.                         (06-07-82)
                        853* ;                                                                  (06-07-82)
0220  7401              854* STRTCLK    MOVEQ     #1,D2                 ;DATA                  (06-07-82)
                        855*                                                                    (09-20-82)
0222  700E              856* ST10       MOVEQ     #STRTSTOP,D0          ;ADDRESS               (09-20-82)
0224  616C              857*            BSR.S     WRITECR               ;                      (06-07-82)
0226  4E75              858*            RTS                             ;                      (06-07-82)
                        859* ;
                        860* ; RDTENTHS - read the tenths register of clock
                        861* ;       EXIT - (NC) = READ OK
                        862* ;              (C)  = ERROR - WRONG ADDRESS
                        863* ;              (D2) = REGISTER VALUE READ
                        864* ;
0228                    865* RDTENTHS
0228  7A03              866*            MOVEQ     #3,D5                 ;CHECK MAX. 4 TIMES FOR CLOCK TURNING
                        867*
022A  7001              868* RDT10      MOVEQ     #TENTHSC,D0           ;read tenth of seconds register
022C  6186              869*            BSR.S     READCR
022E  0C02  000F        870*            CMPI.B    #RDERR, D2            ;do until (no read error) or
0232  56CD  FFF6        871*            DBNE      D5, RDT10             ; (tried 4 times)
                        872*
0236  0C02  0009        873*            CMPI.B    #9, D2                ;if not a BCD digit then wrong address
023A  6204              874*            BHI.S     RDTERR                ;USE OTHER Address
023C  4285              875*            CLR.L     D5
023E  4E75              876*            RTS
                        877*
0240  44FC  0001        878* RDTERR      MOVE.W    #CARRYST, CCR
0244  4E75              879*            RTS
```

```
                        881* ; FINDADDR - FIND ADDRESS OF CHIP'S ADDRESS LATCH. IT IS EITHER RV3ADDR OR RV4ADDR
                        882* ;            DEPENDING ON THE VERSION OF THIS PROCESSOR BOARD.
                        883* ; routine added with 6/7 change.
                        884* ;
0246                    885* FINDADDR
0246  47FA  03A8+       886*              LEA       ADDRREG, A3             ;WHERE TO SAVE CORRECT ADDRESS
024A  43F9  0003  0D01  887*              LEA       RWREG.L,A1
0250  41F9  0003  0F81  888*              LEA       RV4ADDR.L,A0            ;START WITH REV 4 ADDRESS
0256  7801              889*              MOVEQ     #1,D4                   ;TRY ONLY TWO ADDRESSES
                        890*
                        891* ; CHECK IF ADDRESS IN A0 IS CORRECT
                        892* ;
0258                    893* FA10
0258  61CE              894*              BSR.S     RDTENTHS                ;GET STARTING VALUE
025A  651E              895*              BCS.S     FANIT                   ;WRONG CHIP TRY NEXT ADDRESS
025C  0C02  0009        896*              CMPI.B    #9, D2                  ;WAIT UNTIL TENTH OF SECONDS
0260  6706              897*              BEQ.S     FAZERO                  ;IS NEXT TENTH ==) D3.B
0262  1602              898*              MOVE.B    D2,D3
0264  5203              899*              ADDQ.B    #1,D3
0266  6002              900*              BRA.S     FA20
0268  4283              901* FAZERO       CLR.L     D3
026A  3C3C  7530        902* FA20         MOVE.W    #30000,D6               ;MUST READ AT LEAST TENTH SEC.
                        903*
                        904* , READ TENTHS UNTIL IT CHANGES OR UNTIL IT TRIED TO LONG
                        905* ,
026E                    906* FA30
026E  61B8              907*              BSR.S     RDTENTHS                ;GET NEXT VALUE
0270  6508              908*              BCS.S     FANIT                   ;WRONG CHIP TRY NEIT ADDRESS
0272  B602              909*              CMP.B     D2,D3                   ;HAS TIME TICKED
0274  6716              910*              BEQ.S     FAFNDIT                 ;YES, FOUND CORRECT ADDRESS
0276  51CE  FFF6        911*              DBF       D6, FA30                ;READ AGAIN
                        912* ;
                        913* , NOT THIS ADDRESS TRY OTHER ADDRESS
                        914* ;
027A  41F9  0003  0FE1  915* FANIT        LEA       RV3ADDR.L, A0
0280  51CC  FFD6        916*              DBF       D4, FA10
                        917*
                        918* ; ERROR - NEITHER ADDRESS WORKED
                        919* ;
0284  4293              920*              CLR.L     (A3)                    ;SHOW NO CHIP ADDRESS
0286  3E3C  0039        921*              MOVE.W    #CLOCKERR, D7
028A  4E75              922*              RTS
                        923*
                        924* ; FOUND CORRECT ADDRESS
                        925* ;
028C  2688              926* FAFNDIT      MOVE.L    A0, (A3)
028E  4287              927*              CLR.L     D7
0290  4E75              928*              RTS
```

```
                        930* ;
                        931* ;  WRITECR - WRITE A CLOCK REGISTER
                        932* ;           ENTRY : (D0) = REGISTER ADDRESS
                        933* ;                   (D2) = DATA
                        934* ;
0292  6100  FF4A        935* WRITECR     BSR       SELREG                ,DESELECT THEN SELECT REG.
0296  1282              936*             MOVE.B    D2,(A1)               ,WRITE DATA
0298  10BC  0010        937*             MOVE.B    #DSELCT,(A0)          ;DESELECT CHIP
029C  4E75              938*             RTS
                        939* ;
                        940* ; WRITEREGS - WRITE THE CLOCK REGISTERS FROM THE REGISTER ARRAY
                        941* ;
029E  6100  FED0        942* WRITEREGS   BSR       LDADDR                ;GET CHIP ADDRESSES
                        943* ;
                        944* ; WRITE REGISTERS
                        945* ;
02A2  7001              946*             MOVEQ     #1,D0                 ;REGISTER ADDRESS
02A4  45FA  034E+       947*             LEA       REGARRAY,A2
                        948* ,
02A8  141A              949* WRONER      MOVE.B    (A2)+,D2              ;REGISTER DATA
02AA  61E6              950*             BSR.S     WRITECR               ,WRITE DATA
02AC  5200              951*             ADDQ.B    #1,D0                 ;NEXT REGISTER ADDRESS
02AE  0C00  000E        952*             CMPI.B    #STRTSTOP,D0          ,STOP AT START/STOP REG.
02B2  66F4              953*             BNE.S     WRONER
02B4  4E75              954*             RTS
```

```
                        954* ;
                        957* ; TIMWR - SET CLOCK FROM PARAMETER BLOCK
                        958* ;          PARAMETER BLOCK FOR UNITWRITE :
                        959* ;
                        960* ; type WriteClockParameter = record                    range
                        961* ;                              DayofWeek : integer;     1-7
                        962* ;                              Month : integer;         1-12
                        963* ,                              Day : integer;           1-31
                        964* ;                              Hour : integer;          0-23
                        965* ;                              Mins : integer;          0-59
                        966* ;                              Secs : integer;          0-59
                        967* ;                              Tenths : integer;        0-9
                        968* ;                              LeapYear : integer;      0-3
                        969* ;                             end;
                        970* ;
02B4  2641              971* TIMWR      MOVE.L    D1,A3                 ;ADDRESS OF PARAMETER BLOCK
                        972* ;
                        973* ; PROCESS BINARY PARAMETERS
                        974* ,
02B6  611A              975*           BSR.S     VALBIN                ;VALIDATE PARAMS
02BA  6512              976*           BCS.S     TWRERR                ,NO GOOD
02BC  613C              977*           BSR.S     CVTBINR               ;CONVERT BINARY TO BCD OF REGISTERS
                        978* ;
                        979* ; ZERO SECONDS AND TENTHS OF SECONDS                          (09-20-82)
                        980* ;                                                             (09-20-82)
02BE  6100  FEB0        981*           BSR       LDADDR                ; GET CLOCK ADDRESSES  (09-20-82)
02C2  6100  FF58        982*           BSR       STOPCLK               ;                      (09-20-82)
02C6  6100  FF58        983*           BSR       STRTCLK               ;                      (09-20-82)
                        984* ,
                        985* ; WRITE OUT REGISTER ARRAY
                        986* ,
02CA  61D2              987*           BSR.S     WRITEREGS
02CC  4E75              988*           RTS
                        989* ;
                        990* ; ERROR - INVALID CLOCK PARAMETER
                        991* ,
02CE  3E3C  0036        992* TWRERR     MOVE.W    #INVPRM,D7
02D2  4E75              993*           RTS
```

```
                        995* ;
                        996* ; VALBIN - VALIDATE BINARY PARAMETER BLOCK
                        997* ;          ENTRY : (A3) = ADDRESS OF PARAMETER BLOCK
                        998* ;          EXIT  : (NC) = GOOD PARAMETERS
                        999* ;                  (C)  = ERROR, OUT OF RANGE
                       1000* ;
02D4  284B             1001* VALBIN     MOVEA.L   A3,A4                  ,SAVE PB ADDRESS
02D6  7807             1002*            MOVEQ     #NUMWP-1,D4            ,DO ALL 8 PARAMETERS
02D8  4BFA  0246+      1003*            LEA       RANGES,A5              ,LIST OF PARAMETER RANGES(BYTES)
                       1004* ,
                       1005* ; COMPARE EACH PARAMETER TO IT'S LOW AND HI RANGE VALUE
                       1006* ,
02DC  4280             1007* VBCHK      CLR.L     D0
02DE  101D             1008*            MOVE.B    (A5)+,D0               ,GET LOW BOUND RANGE
02E0  B054             1009*            CMP.W     (A4),D0                ,PARAM)=LOW BOUND THEN OK
02E2  6210             1010*            BHI.S     VBERR                  ;ERROR, TO LOW
02E4  101D             1011*            MOVE.B    (A5)+,D0               ;GET HI BOUND RANGE VALUE
02E6  B054             1012*            CMP.W     (A4),D0                ;PARAM(=HI BOUND THEN OK
02E8  650A             1013*            BCS.S     VBERR                  ,ERROR, TO HIGH
02EA  548C             1014*            ADDQ.L    #2,A4                  ,NEXT PARAMETER LOW BYTE
02EC  51CC  FFEE       1015*            DBF       D4,VBCHK
02F0  4280             1016*            CLR.L     D0                     ,SHOW NO ERROR
02F2  4E75             1017*            RTS
                       1018* ,
                       1019* ; ERROR EXIT - OUT OF RANGE
                       1020* ,
02F4  44FC  0001       1021* VBERR      MOVE.W    #CARRYST,CCR           ;SHOW ERROR
02F8  4E75             1022*            RTS
```

```
                        1024* ;
                        1025* ; CVTBINR - CONVERT VALID PARAMETER BLOCK FROM BINARY INTO REGARRAY BCD NIBLES
                        1026* ,          ENTRY . (A3) = ADDRESS OF PARAMETER BLOCK
                        1027* ,
02FA  49FA  0306+       1028* CVTBINR    LEA       HI,A4                 ;HI NIBBLE HOLD ARRAY
02FE  4BFA  030A+       1029*            LEA       LOW,A5                ;LOW NIBBLE HOLD ARRAY
0302  7807               1030*            MOVEQ     #NUMWP-1,D4           ;FOR i := 8 to 1 do
0304  760F               1031*            MOVEQ     #LEAPYR+1,D3          ;INDEX TO PARAMETER
                        1032* ;
0306  4280               1033* CBRNIBS    CLR.L     D0
0308  1033  3000         1034*            MOVE.B    0(A3,D3.W),D0         ;GET PARAMETER(i)
030C  2200               1035*            MOVE.L    D0,D1
030E  82FC  000A         1036*            DIVU      #10,D1                ,HI NIBBLE := PARAM DIV 10
0312  2001               1037*            MOVE.L    D1,D0                 ,LOW IS REMAINDER FROM DIV
0314  4840               1038*            SWAP      D0                    ;LOW := PARAMETER-(HI*10)
0316  1981  4000         1039*            MOVE.B    D1,0(A4,D4.W)         ,SAVE HI
031A  1B80  4000         1040*            MOVE.B    D0,0(A5,D4.W)         ,SAVE LOW
031E  5543               1041*            SUBQ.W    #2,D3                 ,NEXT PARAMETER INDEX
0320  51CC  FFE4         1042*            DBF       D4,CBRNIBS            ,DOWNTO 0
                        1043* ,
                        1044* , SETUP REGISTER ARRAY
                        1045* ,
0324  4283               1046*            CLR.L     D3                    ,REMOVE GARBAGE
0326  47FA  02CC+       1047*            LEA       REGARRAY,A3
032A  780C               1048*            MOVEQ     #RAWRLEN-1,D4         ;MOVE TO 13 REGISTERS
032C  45FA  01E5+        1049*            LEA       NIBBLE,A2             ,WHICH NIBBLE FOR THIS REG.
0330  43FA  01D4+        1050*            LEA       INREGB,A1             ;WHICH PARAMETER IS REG FROM
                        1051* ,
0334  1631  4000         1052* CBRREGS    MOVE.B    0(A1,D4.W),D3         ;INDEX TO HI & LOW FOR THIS REG.
0338  1035  3000         1053*            MOVE.B    0(A5,D3.W),D0         ;ASSUME LOW NIBBLE
033C  4A32  4000         1054*            TST.B     0(A2,D4.W)            ,IF 0 THEN USE LOW NIBBLE
0340  6704               1055*            BEQ.S     CBRULOW               ;IS LOW
0342  1034  3000         1056*            MOVE.B    0(A4,D3.W),D0         ,ELSE GET HI NIBBLE
0346  1780  4000         1057* CBRULOW    MOVE.B    D0,0(A3,D4.W)         ,PUT NIBBLE IN REGISTER HOLD
034A  51CC  FFE8         1058*            DBF       D4,CBRREGS
                        1059* ,
034E  6102               1060*            BSR.S     CVTLPYR               ,CONVERT LEAP YEAR REG
0350  4E75               1061*            RTS
```

```
                        1063* ;
                        1064* ; CVTLPYR - CONVERT LEAP YEAR PARAMETER TO THE REGISTER VALUE FOR THE
                        1065* ;           CLOCK CHIP.  8,4,2,1 WHERE 8 IS FOR LEAP YEAR AND THE OTHER NUMBERS
                        1066* ;           ARE FOR THE YEARS AFTER THE LEAP YEAR.  THEREFORE, 4 IS LEAP YEAR+1,
                        1067* ;           2 IS LEAP YEAR+2, 1 IS LEAP YEAR+3.
                        1068* ;
0352  102B  000C        1069* CVTLPYR     MOVE.B     LYREG-1(A3),D0          ;LEAP YEAR REG. = 13(INDEX=12)
0356  7203              1070*             MOVEQ      #3,D1                   ;D0 IS PARAMETER (RANGE 0-3)
0358  9200              1071*             SUB.B      D0,D1                   ;CALCULATE WHICH BIT TO SET
035A  4280              1072*             CLR.L      D0                      ;BIT# := 3-PARAMETER
035C  03C0              1073*             BSET       D1,D0                   ;D0 IS LEAP YEAR VALUE
035E  1740  000C        1074*             MOVE.B     D0,LYREG-1(A3)          ,PUT IN REGISTER ARRAY
0362  4E75              1075*             RTS
```

```
                        1077* ;
                        1078* ; TIMINST - UNITINSTALL
                        1079* ; INSTALL THE TIMER INTERRUPT ROUTINE AND SET UP THE VIA
                        1080* ;
0364                    1081* TIMINST
                        1082* ;
0364 4DF9 0003 0F7D     1083*           LEA     IER.L,A6
036A 1CBC 007F          1084*           MOVE.B  #DISABL,(A6)            ;TURN OFF ALL INTERUPTS ON VIA
                        1085* ,
                        1086* ; INITIALIZE TIMER TABLE
                        1087* ,
036E 41FA 01C0+         1088*           LEA     TIMERTBL,A0             ;ADDRESS OF TIMER TABLE
0372 43FA 0270+         1089*           LEA     TIMERTBL+(TIMTLEN*NUMENTS),A1 ;1ST BYTE AFTER TABLE
                        1090* ;
0376 30BC 0000          1091* TINST10    MOVE.W  #0,(A0)                 ;CLEAR FLAGS OF EACH ENTRY
037A D0FC 0012          1092*           ADDA.W  #TIMTLEN,A0             ;POINT AT NEXT ENTRY
037E B3C8               1093*           CMPA.L  A0,A1                   ;AT END OF TABLE
0380 66F4               1094*           BNE.S   TINST10                 ;NO
                        1095* ;
                        1096* ; PUT ADDRESS OF INTERRUPT ROUTINE IN VECTOR
                        1097* ;
0382 41FA FC7C+         1098*           LEA     TIMINT,A0
0386 21C8 0074          1099*           MOVE.L  A0,VECTOR.W
                        1100* ,
                        1101* ; SETUP VIA
                        1102* ;
038A 13FC 0040 0003     1103*           MOVE.B  #ACRBYTE,ACR.L          ;FREE RUN MODE PB7 OUTPUT DISABLED
0390 0F77
0392 13FC 0050 0003     1104*           MOVE.B  #TIMEL,T1LL.L           ;TIMER #1 LATCH LOW
0398 0F6D
039A 13FC 00C3 0003     1105*           MOVE.B  #TIMEH,T1LH.L           ;TIMER #1 LATCH HIGH
03A0 0F6F
03A2 13FC 00C3 0003     1106*           MOVE.B  #TIMEH,T1CH.L           ;TIMER #1 COUNTER HIGH - FORCE LOAD
03A8 0F6B
03AA 13FC 00FF 0003     1107*           MOVE.B  #CLEAR,IFR.L            ;CLEAR IFR
03B0 0F7B
                        1108* ;
                        1109* ; ENABLE TIMER #2
                        1110* ;
03B2 1CBC 00C0          1111*           MOVE.B  #ENBLT1,(A6)            ;TURN INTERUPTS ON FOR T1
                        1112* ;
                        1113* ; INITIALIZE CLOCK - SOURCE IN TIMER.CLK.TEXT INCLUDE FILE
                        1114* ;
03B6 6000 FE30          1115*           BRA     INITCLK                 ;DOES RETURN WHEN INITCLK DOES
```

```
                        1117* ;
                        1118* ; TIMUNMT - UNITUNMOUNT
                        1119* ; TURN OFF THE VIA INTERRUPTS AND POINT THE TIMER INTERRUPT VECTOR AT A RTE.
                        1120* ;
03BA                    1121* TIMUNMT
03BA  13FC  007F 0003   1122*           MOVE.B    #DISABL,IER.L          ,TURN OFF ALL INTERUPTS ON VIA
03C0  0F7D
03C2  41FA  FC96+       1123*           LEA       TUNRTE,A0              ,WITH TIMER INTERRUPT CODE
03C6  21C8  0074        1124*           MOVE.L    A0,VECTOR.W            ,POINT VECTOR AT RTE
03CA  4E75              1125*           RTS
```

```
                        1127* ,
                        1128* , TIMST - UNITSTATUS
                        1129* , THIS PPROCEDURE CONTAINS THE BELL ROUTINE AND THE 4 TIMER TABLE MANIPULATION
                        1130* , PROCEDURES, CREATE, DELETE, DISABLE, AND ENABLE.
                        1131* ;
                        1132* ,           ENTRY .     D2 - CONTROL CODE USED TO SELECT FUNCTIONS
                        1133* ,                       A3 - BUFFER ADDRESS = PTR TO PARAMETER BLOCK
                        1134* ,
03CC  0C42  0004        1135* TIMST       CMPI.W      #ENABLEC,D2             ;VALID FUNCTION CODE
03D0  6212              1136*             BHI.S       TSTERR                  ,NO
                        1137* ,
03D2  45FA  015C+       1138*             LEA         TIMERTBL,A2             ;ADDRESS OF TIMER TABLE
03D6  43FA  0012+       1139*             LEA         TSTTBL,A1               ,TURN THE CONTROL CODE INTO A
03DA  E34A              1140*             LSL.W       #1,D2                   ,INDEX TO THE FUNCTION
03DC  3431  2000        1141*             MOVE.W      0(A1,D2.W),D2
03E0  4EF1  2000        1142*             JMP         0(A1,D2.W)              ;DO FUNCTION
                        1143* ;
                        1144* , Invalid Function Code Error
                        1145* ,
03E4  3E3C  0038        1146* TSTERR      MOVE.W      #INVFNC,D7
03E8  4E75              1147*             RTS
                        1148* ,
                        1149* , THE TIMER DRIVER JUMP TABLE
                        1150* ,
03EA  00A0              1151* TSTTBL      DATA.W      TSTBELL-TSTTBL          ;BELL
03EC  000A              1152*             DATA.W      TSTCRE8-TSTTBL          ;CREATE TABLE ENTRY
03EE  004E              1153*             DATA.W      TSTDELT-TSTTBL          ;DELETE TABLE ENTRY
03F0  0060              1154*             DATA.W      TSTDSBL-TSTTBL          ;DISABLE TABLE ENTRY
03F2  0072              1155*             DATA.W      TSTENBL-TSTTBL          ;ENABLE TABLE ENTRY
```

```
                    1157* ;
                    1158* ; TSTCRE8 - CREATE TABLE ENTRY
                    1159* ,          ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                    1160* ,                  A2 = ADDRESS OF TIMER TABLE
                    1161* ,                  A4 = VALUE WHEN TIMER DRIVER CALLED
                    1162* ;                  A5 = VALUE WHEN TIMER DRIVER CALLED
                    1163* ,          PARAMETER BLOCK .
                    1164* ;            1) ADDRESS OF USER SERVICE ROUTINE TO INSTALL IN ENTRY (LONGWORD)
                    1165* ;            2) COUNT OF 50 MILLISECOND PERIODS TO WAIT (WORD)
                    1166* ,            3) FLAGS (WORD) -
                    1167* ;                      bit D1 = CONTINUOUS/1SHOT MODE FLAG
                    1168* ;                      bit D2 = SKIP FIRST CALL FLAG
                    1169* ;            4) RETURN SPACE FOR TABLE ENTRY ID, THE ENTRY NUMBER (WORD)
                    1170* ;
03F4  4280          1171* TSTCRE8     CLR.L     D0                        ;ENTRY #
03F6  4283          1172*             CLR.L     D3                        ;ENTRY INDEX
                    1173* ;
                    1174* , FIND AN UNUSED ENTRY IF ONE AVAILABLE
                    1175* ;
03F8  0832  0000 3001 1176* TCRCKNXT   BTST      #VALIDENT,TFLAGS+1(A2,D3.W)
03FE  6712          1177*             BEQ.S     TCRFOUND                  ,FOUND ONE
0400  0643  0012    1178*             ADDI.W    #TIMTLEN,D3               ,ELSE SEE IF AT END OF TABLE
0404  5240          1179*             ADDQ.W    #1,D0                     ,NEXT ENTRY NUMBER
0406  0C40  000A    1180*             CMPI.W    #NUMENTS,D0               ,IN TABLE?
040A  66EC          1181*             BNE.S     TCRCKNXT                  ,YES
                    1182* ,
                    1183* ; ERROR TABLE FULL
                    1184* ,
040C  3E3C  0033    1185*             MOVE.W    #TBLFULL,D7
0410  6024          1186*             BRA.S     TCREXIT
                    1187* ;
                    1188* ; FOUND UNUSED ENTRY - SET IT UP
                    1189* ;
0412  259B  3002    1190* TCRFOUND    MOVE.L    (A3)+,PTRUSRTN(A2,D3.W)   ;PUT IN USER SERVICE RTN ADDRES
0416  3593  3006    1191*             MOVE.W    (A3),TCOUNT(A2,D3.W)      ,COUNT OF 50 MS. TICKS
041A  359B  3008    1192*             MOVE.W    (A3)+,TDWNCNT(A2,D3.W)    ,SET DOWN COUNTER
041E  321B          1193*             MOVE.W    (A3)+,D1                  ,GET FLAGS
0420  08C1  0000    1194*             BSET      #VALIDENT,D1              ;SHOW ENTRY IN USE
0424  0881  0003    1195*             BCLR      #ENBLDSBL,D1              ,SHOW ENABLED
0428  3581  3000    1196*             MOVE.W    D1,TFLAGS(A2,D3.W)        ;PUT IN ENTRY
042C  258C  300A    1197*             MOVE.L    A4,REGA4(A2,D3.W)         ;SAVE USERS A4 AND A5 REGISTERS
0430  258D  300E    1198*             MOVE.L    A5,REGA5(A2,D3.W)
                    1199* ;
                    1200* ; RETURN TO USER TABLE ENTRY ID (THE ENTRY NUMBER)
                    1201* ;
0434  3680          1202*             MOVE.W    D0,(A3)
0436  4E75          1203* TCREXIT     RTS
```

```
                        1205* ,
                        1206* , TSTDELT - DELETE TABLE ENTRY
                        1207* ,           ENTRY . A3 = ADDRESS OF PARAMETER BLOCK
                        1208* ,                  A2 = ADDRESS OF TIMER TABLE
                        1209* ,           PARAMETER BLOCK .
                        1210* ,              1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                        1211* ,
0438                    1212* TSTDELT                                          ,GET INDEX TO ENTRY
0438  613A              1213*            BSR.S     VALIDID                     ;IS ID VALID?
043A  6508              1214*            BCS.S     TDELERR                     ,INVALID-ERROR EXIT
                        1215* ,
                        1216* , VALID ENTRY INDEX - DELETE ENTRY
                        1217* ,
043C  08B2  0000  3001  1218*            BCLR      #VALIDENT,TFLAGS+1(A2,D3.W)
0442  6004              1219*            BRA.S     TDELEXIT
                        1220* ,
                        1221* , INVALID TABLE ID ERROR
                        1222* ,
0444  3E3C  0032        1223* TDELERR    MOVE.W    #INVTBLID,D7
0448  4E75              1224* TDELEXIT   RTS
```

```
                        1226* ;
                        1227* ; TSTDSBL - DISABLE TABLE ENTRY
                        1228* ;          ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                        1229* ;                  A2 = ADDRESS OF TIMER TABLE
                        1230* ;          PARAMETER BLOCK :
                        1231* ;             1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                        1232* ;
044A                    1233* TSTDSBL                                        ;GET INDEX TO ENTRY
044A  6128              1234*            BSR.S     VALIDID                   ;IS ID VALID?
044C  6508              1235*            BCS.S     TDSBERR                   ;INVALID-ERROR EXIT
                        1236* ;
                        1237* ; VALID ENTRY INDEX - DISABLE ENTRY
                        1238* ;
044E  08F2  0003  3001  1239*            BSET      #ENBLDSBL,TFLAGS+1(A2,D3.W)
0454  6004              1240*            BRA.S     TDSBEXIT
                        1241* ;
                        1242* ; INVALID TABLE ID ERROR
                        1243* ;
0456  3E3C  0032        1244* TDSBERR     MOVE.W    #INVTBLID,D7
045A  4E75              1245* TDSBEXIT    RTS
```

```
                    1247* ,
                    1248* ; TSTENBL - ENABLE TABLE ENTRY
                    1249* ,           ENTRY . A3 = ADDRESS OF PARAMETER BLOCK
                    1250* ,                   A2 = ADDRESS OF TIMER TABLE
                    1251* ,           PARAMETER BLOCK :
                    1252* ,              1) TABLE ENTRY ID, ENTRY # TO ENTRY (WORD)
                    1253* ;
045C                1254* TSTENBL                                       ;GET INDEX TO ENTRY
045C 6116           1255*           BSR.S     VALIDID                   ;IS ID VALID?
045E 650E           1256*           BCS.S     TENBERR                   ;INVALID-ERROR EXIT
                    1257* ,
                    1258* , VALID ENTRY INDEX - ENABLE ENTRY AND RESTART DOWN COUNTER
                    1259* ,
0460 35B2 3006 3008 1260*           MOVE.W    TCOUNT(A2,D3.W),TDWNCNT(A2,D3.W)
0466 08B2 0003 3001 1261*           BCLR      #ENBLDSBL,TFLAGS+1(A2,D3.W)
046C 6004           1262*           BRA.S     TENBEXIT
                    1263* ,
                    1264* , INVALID TABLE ID ERROR
                    1265* ,
046E 3E3C 0032      1266* TENBERR   MOVE.W    #INVTBLID,D7
0472 4E75           1267* TENBEXIT  RTS
                    1268* ;
                    1269* , VALIDID - VALIDATE TABLE ENTRY ID IN PARAMETER
                    1270* ;           ENTRY . A3 = ADDRESS OF PARAMETER BLOCK
                    1271* ,           EXIT  . D3 = TABLE INDEX
                    1272* ,                    (C) = INVALID TABLE ENTRY ID
                    1273* ,                    (NC)= VALID TABLE ENTRY ID
                    1274* ,
0474 3613           1275* VALIDID   MOVE.W    (A3),D3                   ;GET TABLE ENTRY ID
                    1276* ;
                    1277* , TABLE ENTRY ID IS THE ENTRY NUMBER - MAKE SURE LESS THAN NUMBER OF ENTRIES IN TABLE
                    1278* ,
0476 0C43 0009      1279*           CMPI.W    #NUMENTS-1,D3             ;IS INDEX LESS THAN TABLELN?
047A 6306           1280*           BLS.S     VALCALC                   ;YES, CALCULATE INDEX
                    1281* ,
047C 44FC 0001      1282* VALERR    MOVE.W    #CARRYST,CCR              ;SHOW ERROR
0480 6006           1283*           BRA.S     VALEXIT
                    1284* ,
                    1285* , HAVE VALID TABLE ENTRY
                    1286* ,
0482 C6FC 0012      1287* VALCALC   MULU      #TIMTLEN,D3               ;CALCULATE INDEX
0486 4280           1288*           CLR.L     D0                        ;CLEAR CARRY
0488 4E75           1289* VALEXIT   RTS
```

```
                    1291* , TSTBELL - BELL ROUTINE
                    1292* ,          ENTRY : A3 = ADDRESS OF PARAMETER BLOCK
                    1293* ;          PARAMETER BLOCK :
                    1294* ,            1) FREQUENCY (WORD)
                    1295* ;            2) SPEAKER ON/OFF PATTERN (BYTE)
                    1296* ,            3) FILLER (BYTE)
                    1297* ;            4) DURATION IN 50 MILLISECOND PERIODS (WORD)
                    1298* ; INIT VIA FOR FREQUENCY W/O DISTURBING TIMER #1
                    1299* ;
048A 4BF9 0003 0F75 1300* TSTBELL    LEA       SHIFTREG.L,A5
0490 1ABC 0000      1301*           MOVE.B    #0,(A5)                 ;TURNOFF BELL FOR SURE
0494 0039 0010 0003 1302*           ORI.B     #RUNT2,ACR.L            ;SET TIMER #2 AS COUNT DOWWN
049A 0F77
049C 613A           1303*           BSR.S     SETT2                   ;PUT FREQUENCY IN TIMER
049E 49FA 0144+     1304*           LEA       IFLAGS,A4
04A2 0894 0000      1305*           BCLR      #SHUTOFF,(A4)           ;PUT TIMER ON
                    1306* ;
                    1307* , CALL CREATE TO SETUP ONE SHOT INTERVAL TIMER CALL
                    1308* ;
04A6 43FA 013E+     1309*           LEA       IPRMBLK,A1              ,ADDR OF INTERNAL PARM BLOCK
04AA 2F0B           1310*           MOVE.L    A3,-(SP)                ;SAVE PARAMETER BLOCK ADDRESS
04AC 302B 0004      1311*           MOVE.W    DURATN(A3),D0           ;SAVE THE COUNT
04B0 2649           1312*           MOVEA.L   A1,A3                   ;CREATE EXPECTS PRM BLK ADR IN A3
04B2 41FA 0034+     1313*           LEA       BELSRVR,A0              ;BELL SERVICE ROUTINE ADDRESS
04B6 22C8           1314*           MOVE.L    A0,(A1)+                ,PUT IN PARAMETER BLOCK
04B8 3280           1315*           MOVE.W    D0,(A1)                 ;PUT IN COUNT
04BA 6100 FF38      1316*           BSR       TSTCRE0                 ;CALL CREATE
04BE 265F           1317*           MOVE.L    (SP)+,A3                ;BELL PARAMETER BLOCK ADDRESS
                    1318* ;
04C0 1AAB 0002      1319*           MOVE.B    PATTERN(A3),(A5)        ;TURN ON BELL
                    1320* ;
                    1321* ; WAIT FOR SHUT OFF
                    1322* ;
04C4 0814 0000      1323* TBELWAIT   BTST      #SHUTOFF,(A4)           ;DONE?
04C8 67FA           1324*           BEQ.S     TBELWAIT                ;NO
                    1325* ;
                    1326* ; DONE SHUT OFF TIMER #2 AND BELL
                    1327* ;
04CA 1ABC 0000      1328* TBELDONE   MOVE.B    #0,(A5)                 ,CLEAR SHIFT REG TO SHUT OFF BELL
04CE 0239 00EF 0003 1329*           ANDI.B    #STOPT2,ACR.L
04D4 0F77
04D6 4E75           1330* TBELEXIT   RTS
```

```
                    1332* ,
                    1333* , SETT2 - SET TIMER #2 TO FREQUENCY IN PARAMETER BLOCK
                    1334* ,
04D8  13EB  0001  0003  1335* SETT2       MOVE.B    1(A3),T2LL.L            ,SET LATCH
04DE  0F71
04E0  13D3  0003  0F73  1336*             MOVE.B    (A3),T2CH.L             ;SET COUNTER AND CLEAR IFR T2 FLAG
04E4  4E75              1337*             RTS
                    1338* ;
                    1339* ; BELL TIMER SERVICE ROUTINE
                    1340* ;
04E8  41FA  00FA+       1341* BELSRVR     LEA       IFLAGS,A0               ,TELL BELL ROUTINE DONE
04EC  08D0  0000        1342*             BSET      #SHUTOFF,(A0)           ,& TO SHUT OFF SPEAKER AND
04F0  4E75              1343*             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     4,5,5,4,4,3,3,2,2,0,1,1,7 ;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           ; 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           , 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           ; 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           , 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           , 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           , 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           , 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
      000000B4        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  C0C00000        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
      0C000000C       1398* RARDLEN   EQU       %-REGARRAY                ,NUMBER OF REGISTERS READ
0600  00              1399*           DATA.B    0
      0C000000D       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   00000001   DSELCT     00000010
 ACTIVE    00000006   CARRYST   00000001   CONTISHT 00000001   CVOEXIT    0001A0+  DURATN     00000004
 ADDRREG     0005F0+  CBCDBIN     0001A2+  CSATTR1  00000010   CVTBINR    0002FA+  ENABLEC    00000004
 BELSRVR     0004E8+  CBRNIBS     000306+  CSATTR2  00000011   CVTLPYR    000351+  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  00000000   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	TICHKNIT	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	0006BE+
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	000000SA	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	000D5C+
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	TCRCKNIT	0003F8+	UTRO	0000001A
IPRMBLK	0005E6+	REGA5	0000000E	STNDRV	00000002	TCREXIT	000436+	UTSIZ	00000010
LDADDR	000170+	REGARRAY	0005F4+	STNMBR	00000000	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	00040A+	RWREG	00030D01	STRTSTOP	0000000E	TDSBEXIT	00045A+	UTTYP	00000017
LYREG	0000000D	SCBOOTDV	00000036	STTYPE	00000001	TDWNCNT	00000008	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*
CONTISHT    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*
CVTBINB     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
ENBLTI      320* 1111
FA10        893*  916
FA20        900   902*
```

```
  1* , file .  drv.dtacom.text
  2* , date :  09-Feb-1983
  3* ,
  4* ; This is the datacom driver source
  5* ,
  6* ;     date          by          rev level             comments
  7* ;_____________________________________________________________________
  8* ;
  9* ,   10/12/82       rpk          1            initial version    no protocols-nothin
 10* ,   10/21/82       rpk          2            added auto line feed flag on writes
 11* ,   10/27/82       rpk          3            MADE BUSY ONLY TELL ABOUT READ BUFFER
 12* ,   11/29/82       kb           4a           Started adoitions for protocols and
 13* ,                                              compatibility with old printer driver
 14* ,                                              unitstatus interface.
 15* ,   01/05/83       kb           4e           changed auto line feed flag useage
 16* ;   01/06/83       kb           4f           added switch of water marks when switch back
 17* ,                                              to default read buffer.
 18* ;   01/12/83       kb                        fixed bug in FINDLIN routine, using
 19* ;                                              wrong register for bit test. changed
 20* ,                                              bit number register from D0 to D5.
 21* ;   02/09/83       kb                        added setup of UART base reg in DCTLINT
 22* ;                                                in ChkLines
 23* ;
 24* ;*************************************************************************************
 25* ;
 26* ; INCLUDE FILES USED :
 27* ;       /ccos/os.gbl.asm.text                   ;OS GLOBAL EQUATES
 28* ;       dcom.equ.text                           ;definitions for driver
 29* ;
 30* ; INCLUDE OS GLOBALS HERE
285*             LIST       1
286*             INCLUDE    'DCOM.EQU.TEXT'
```

```
                        288* , Equates for Data com driver
                        289* ; file . DCOM.EQU.TEXT
                        290* , date : 24 - January - 1983
                        291* ,
                        292* ;added definition of LFsprsflg as byte value of UCSD and Apple p-systems
                        293* ,      mode flags (D5) : 1-24-83  kb
                        294* ,
                        295* ,
                        296* ; EQUATES FOR ALL DATACOM DRIVER SOFTWARE
                        297* , BIT NUMBER DEFINITIONS
                        298* ;
00000000                299* BITD0       EQU        0                       ,BIT 0
00000001                300* BITD1       EQU        1                       ;BIT 1
00000002                301* BITD2       EQU        2                       ,BIT 2
00000003                302* BITD3       EQU        3                       ,BIT 3
00000004                303* BITD4       EQU        4                       ,BIT 4
00000005                304* BITD5       EQU        5                       ,BIT 5
00000006                305* BITD6       EQU        6                       ,BIT 6
00000007                306* BITD7       EQU        7                       ,BIT 7
                        307* ,
                        308* ; Flags for port common Flag word
                        309* ,
00000000                310* PORTFLG      EQU        BITD0                   ,IF 0 THEN INIT PORT 0 ELSE INIT PORT 1
                        311* ,
                        312* ; Buffer control table INTERNAL Flag  bit definitions **LO BYTE***  BF_INTL
                        313* , Low order byte
                        314* ,
00000004                315* ENQFLG      EQU        BITD4                   ,SENT ENQ WAITING FOR ACK
                        316* ,
                        317* , Buffer Control Table PROTOCOL flag bit definitions  ****lo byte**** BF_PROF
                        318* ,
00000000                319* LINE        EQU        BITD0                   ,LINE TYPE HANDSHAKE
00000001                320* XONXOFF     EQU        BITD1                   ,XON/XOFF HANDSHAKE
00000002                321* ENQACK      EQU        BITD2                   ,ENQ/ACK HANDSHAKE
00000003                322* CTSLIN      EQU        BITD3                   ;LINE IS CTS
00000004                323* DSRLIN      EQU        BITD4                   ,LINE IS DSR
00000005                324* DCDLIN      EQU        BITD5                   ,LINE IS DCD
00000006                325* INVBUSY     EQU        BITD6                   ,1=LINE IS INVERTED(0) WHEN BUSY
00000007                326* ETXACK      EQU        BITD7                   ,ETX/ACK HANDSHAKE
                        327* ,
                        328* ; BUFFER CONTROL TABLE PROTOCOL FLAG BIT DEFINITIONS ***HI BYTE***BF_PROF
                        329* ,
00000000                330* PROT_F2     EQU        BITD0                 ,IF SET THEN SOME TYPE OF PROTOCOL EXISTS
                        331*                                              ,ELSE NO PROTOCOLS --BUFFERS OVERFLOW ETC
00000001                332* MODM_F2     EQU        BITD1                 ,IF SET THEN A MODEM PROTOCOL EXISTS
00000002                333* NMOD_F2     EQU        BITD2                 ;IF SET THEN NULL MODEM PROTOCOL(PROBABLY OF
LITTLE USE)
00000003                334* FULL_F2     EQU        BITD3                 ,IF SET THEN FULL DUPLEX (DEFAULT)
                        335* ,                                            ,OTHERWISE HALF DUPLEX
                        336* ;
                        337* ;
                        338* ;      WRITE BUFFER flag word bit definitions    FLAG 1 ->>lo byte
                        339* ;
00000000                340* BUSY_W1     EQU        BITD0                   ,WRITE BUSY FLAG
00000001                341* ERR_W1      EQU        BITD1                   ,BUFFER SIZE ERROR FOUND IN XMIT INT RTN
```

```
00000002        342* ALTBF_W1    EQU     BITD2           ;IF SET(1) HAVE ALTERNATE BUFFER TO USE
00000004        343* OUTE_W1     EQU     BITD4           ;IF 0 DATA FROM BUFFER TO PORT ENABLED
00000005        344* INPE_W1     EQU     BITD5           ;IF 0 DATA FROM USER TO BUFFER ENABLED
00000006        345* OUTC_W1     EQU     BITD6           ;IF SET(1) THEN USER IS CONTROLLING OUTE
                346*                                     ;OTHERWISE CONTROLLED INTERNALLY
00000007        347* INPC_W1     EQU     BITD7           ;IF SET(1) THEN USER IS CONTROLLING INPE
                348* ,
                349* ;     WRITE BUFFER flag word bit definitions    FLAG 2 ->>lo byte
                350* ,
00000000        351* FULL_W2     EQU     BITD0           ;IF SET (1) THEN BUFFER IS FULL
00000001        352* EMPT_W2     EQU     BITD1           ;IF SET (1) THEN BUFFER IS EMPTY
00000002        353* LOST_W2     EQU     BITD2           ;DATA LOST ON INPUT (USER OVERRUNS BUFFER)
00000003        354* SNDLF_W2    EQU     BITD3           ;IF SET THEN SEND AN LF
00000004        355* AULF_W2     EQU     BITD4           ;IF SET THEN always send a LF after a CR
                356* ,
                357* ;     READ BUFFER flag word bit definitions     FLAG 1 ->>LO BYTE
                358* ;
00000000        359* BUSY_R1     EQU     BITD0           ;READ BUSY FLAG
00000001        360* ERR_R1      EQU     BITD1           ;UART ERROR FLAG
00000002        361* ALTBF_R1    EQU     BITD2           ;IF SET(1) HAVE ALTERNATE BUFFER TO USE
00000004        362* OUTE_R1     EQU     BITD4           ;IF 0 DATA FROM BUFFER TO USER ENABLED
00000005        363* INPE_R1     EQU     BITD5           ;IF 0 DATA FROM PORT TO BUFFER ENABLED
00000006        364* OUTC_R1     EQU     BITD6           ;IF SET(1) THEN USER IS CONTROLLING OUTE
                365*                                     ;OTHERWISE CONTROLLED INTERNALLY
00000007        366* INPC_R1     EQU     BITD7           ;IF SET(1) THEN USER IS CONTROLLING INPE
                367* ,
                368* ;     READ BUFFER flag word bit definitions   FLAG 2 LO BYTE
                369* ,
00000000        370* FULL_R2     EQU     BITD0           ;IF SET (1) THEN BUFFER IS FULL
00000001        371* EMPT_R2     EQU     BITD1           ;IF SET (1) THEN BUFFER IS EMPTY
00000002        372* LOST_R2     EQU     BITD2           ;DATA LOST ON INPUT (PORT OVERRUNS BUFFER)
                373* ,
                374* ;     CONTROL CHARACTER BUFFER flag word bit definitions  LO BYTE
                375* ,
00000000        376* FULL_CB     EQU     BITD0           ;IF SET (1) THEN BUFFER IS FULL
00000001        377* EMPT_CB     EQU     BITD1           ;IF SET (1) THEN BUFFER IS EMPTY
                378* ;
                379* ; 68000 Interrupt Auto Vector Addresses
                380* ;
00000064        381* VEC1        EQU     $64             ;AUTO VECTOR #1-DATA COM CONTROL
                382*                                     ;This is the VIA used in line
                383*                                     ;protocols
00000068        384* VEC2        EQU     $68             ;AUTO VECTOR #2-DC 1
00000070        385* VEC4        EQU     $70             ;AUTO VECTOR #4-DC 0
                386* ,
                387* ;
                388* ;*************************************************************************
                389* ;
                390* ; Unit I/O Command codes  --found IN D4.W
                391* ,
00000000        392* INSTCMD     EQU     0               ; Install the unit
00000001        393* READCMD     EQU     1               ; read command
00000002        394* WRCMD       EQU     2               ; write command
00000003        395* CLRCMD      EQU     3               ; CLEAR THE UNIT
```

```
00000004          396* BUSYCMD     EQU       4                    ; busy command
00000005          397* ST$CMD      EQU       5                    ; STATUS COMMAND -ACTUAL COMMANDS IN D2"W
00000006          398* UNMCMD      EQU       6                    ; unmount command
                  399* ;
                  400* ; mode flags -- found in D4.W
                  401* ;
0000000C          402* LFsprsflg   EQU       $0C                  ; Auto LF suppress bits 2 or 3
                  403* ;
                  404* ;   STATUS COMMANDS FOUND IN D2.W
                  405*
                  406* ; functions compatible with old printer driver
                  407*
00000000          408* D2_FREEW    EQU       0                    ;RETURN WRITE BUFFER FREE SPACE
00000001          409* D2_BAUDS    EQU       D2_FREEW+1           ;SET READ/WRITE BAUD RATE
00000002          410* D2_PARTY    EQU       D2_BAUDS+1           ;SET PARITY
00000003          411* D2_FREER    EQU       D2_PARTY+1           ;RETURN READ BUFFER FREE SPACE
00000004          412* D2_CHARS    EQU       D2_FREER+1           ;SET CHARACTER SIZE
00000005          413* D2_HANDS    EQU       D2_CHARS+1           ;SET HANDSHAKE METHOD
00000006          414* D2_BFCTRL   EQU       D2_HANDS+1           ;RETURN STATE OF BUFFER CONTROL TABLE
                  415*
                  416* ; new functions
                  417*
00000007          418* D2_RESTS    EQU       D2_BFCTRL+1          ;RETURN READ STATUS
00000008          419* D2_WRSTS    EQU       D2_RESTS+1           ;RETURN WRITE STATUS
00000009          420* D2_REAHI    EQU       D2_WRSTS+1           ,SET READ HI WATER MARK(NMBR OF CHARS LEFT F
REE)
0000000A          421* D2_REALO    EQU       D2_REAHI+1           ;SET READ LO WATER MARK(NMBR OF CHARACTERS L
FT IN BFR)
0000000B          422* D2_OUTRD    EQU       D2_REALO+1           ;USER DISABLE OF OUTBOUND READ (BUFFER DISAB
LE)
0000000C          423* D2_INBRD    EQU       D2_OUTRD+1           ,USER DISABLE OF INBOUND READ  (DEVICE DISAB
LE)
0000000D          424* D2_OUTWT    EQU       D2_INBRD+1           ;USER DISABLE OF OUTBOUND WRITE (DEVICE DISA
BLE)
0000000E          425* D2_INBWT    EQU       D2_OUTWT+1           ;USER DISABLE OF INBOUND WRITE (BUFFER DISAB
LE)
0000000F          426* D2_WBCHR    EQU       D2_INBWT+1           ;RETURN THE NUMBER OF CHARACTERS IN WRITE BU
FFER
00000010          427* D2_RBCHR    EQU       D2_WBCHR+1           ;RETURN THE NUMBER OF CHARACTERS IN READ BUF
FER
00000011          428* D2_ATLF     EQU       D2_RBCHR+1           ;TOGGLE auto LineFeed flag
00000012          429* D2_ENQ      EQU       D2_ATLF+1            ;SET number of chars between ENQ's or ETX's
00000013          430* D2_RDALTB   EQU       D2_ENQ+1             ;Set Read Alternate Buffer
00000014          431* D2_WTALTB   EQU       D2_RDALTB+1          ,Set Write Alternate Buffer
00000014          432* TBLSTATE    EQU       D2_WTALTB            ;The last status function code
                  433* ;
                  434* ;
                  435* ;          THE ABOVE IS WILD AND WOOLY AND MAY BE OF LITTLE US TO A SIMPLE
                  436* ;          HIGHER LEVEL PROTOCOL-- HOWEVER THE HIGHER YOU GET THE MORE USE
                  437* ;          SOME OF THESE REPORTING FUNCTIONS MAY BE
                  438* ;
                  439* ;
                  440* ;
                  441* ;*************************************************************************
```

```
                        443* ,
                        444* , 68000 status Register values
                        445* ,
                        446* ,
00000000                447* UPRMSK      EQU     $A000           , KEEPS ALL STATES AND TRACE BITS AS IS
00002000                448* STATEMSK    EQU     $2000           , ANDS OFF STATE BIT
00008000                449* TRACEMSK    EQU     $8000           ; ANMDS OFF TRACE BIT
00000700                450* INTMSK      EQU     $0700           , ANDS OFF ALL INT LEVELS
00000400                451* INT4        EQU     $400            , INTERRUPT LEVEL 4 AND LOWER
00000200                452* INT2        EQU     $200            ; ETC LEVEL2
00000100                453* INT1        EQU     $100            ;ETC LEVEL 1
00002400                454* DISINT4     EQU     $2400           , Disable all DataCom 0 (priority 4) and be
low device ints
00002200                455* DISINT2     EQU     $2200           , DISABLE DATACOM 1 (priority 2 )and below
ints
00002100                456* DISINT1     EQU     $2100           , Disable DataCom-Control int
00000001                457* CARRYST     EQU     $0001           , CCR with carry set
                        458* ,
                        459* , VIA Addresses
                        460* ,
00030F63                461* ORA         EQU     $30F63          ; PORT A
00030F67                462* DDRA        EQU     $30F67          ; PORT A DATA DIRECTION REG
00030F7F                463* NHIRA       EQU     $30F7F          , PORT A W/O HANDSHAKE(IGNORE DDRA)
                        464* ,
                        465* ; VIA register values
                        466* ,
00000080                467* IODDRA      EQU     $80             ; PORT A BIT CONFIGURATION
                        468* ,
                        469* ,**********************************************************************
                        470* ,
                        471* ; UART register definitions
                        472* ,
00030F20                473* UARTDC0     EQU     $30F20          ;BASE ADDRESS OF DATACOM 0 UART
00000020                474* DC1OFF      EQU     $20             ;OFFSET FROM DC0 BASE TO DC1 BASE
00000001                475* DATAREG     EQU       1             ;DATA PORT REGISTER INDEX
00000003                476* STATRI      EQU       3             ;STATUS REGISTER INDEX
00000005                477* CMDREGI     EQU       5               ;COMMAND REGISTER INDEX
00000007                478* CTLREGI     EQU       7               ;CONTROL REGISTER INDEX
                        479* ;
                        480* ;   UART STATUS REGISTER EQUATES
                        481* ;
00000000                482* S_PARI      EQU     BITD0           ;PARITY ERROR IF SET--SELF CLEARING
00000001                483* S_FRAME     EQU     BITD1           ;FRAMING ERROR IF SET --SELF CLEARING
00000002                484* S_OVRN      EQU     BITD2           ;DATA OVERRUN IF SET
00000003                485* S_RCVF      EQU     BITD3           ;RECEIVE REGISTER FULL IF SET -CLEARED BY RE
AD DATA
00000004                486* S_WRTE      EQU     BITD4           ;WRITE REGISTER EMPTY IF SET
00000005                487* S_DCD       EQU     BITD5           ;DATA CARRY DETECT IF LO ---WIRED LOW
00000006                488* S_DSR       EQU     BITD6           ;DATA SET READY IF LOW --- WIRED LOW
00000007                489* S_IRQ       EQU     BITD7           ;INTERRUPT REQUEST IF SET
                        490* ;                                 S_RCVF EQUIVALENT TO RCVBF
                        491* ,                                 S_WRTE EQUIVALENT TO XMITBE
                        492* ,
                        493* ; UART status registe masks
                        494* ;
00000007                495* S_ERRbits   EQU     $07             ;Parity, Framing, and Overrun
                        496* ;
```

```
                        497* ;   UART COMMAND REGISTER
                        498* ;                    NOTE:cannot or members of same section together
                        499* ;
            00000000    500* CM_DISP    EQU     0                   ;DISABLE PARITY
            00000020    501* CM_OPBT    EQU     $20                 ;ODD PARITY BOTH XMIT AND RCV
            00000040    502* CM_EPBT    EQU     $40                 ;EVEN PARITY BOTH XMIT AND RECEIVE
            000000A0    503* CM_MPBD    EQU     $A0                 ;MARK PARITY BIT UPON XMIT -PARITY CK DISABL
ED
            000000E0    504* CM_SPBD    EQU     $E0                 ;SPACE PARITY BIT ON XMIT - PARITY CK DISABL
ED
                        505* ;-------------------------------------
            00000010    506* CM_ECHO    EQU     $10                 ;IF SET-ECHO MODE FOR RECEIVER
                        507* ;-------------------------------------
            00000001    508* CM_DTRL    EQU     $1                  ;ENABLE RCVR/XMITRR IF SET DTR BAR=LOW
                        509* ;-------------------------------------
            00000002    510* CM_IRQD    EQU     $2                  ;DISABLE INTERRUPTS IF SET --- NOTE CORVUS C
UTEY
                        511*                                        ;  THIS IS ENABLED FROM STATUS BIT 3, NOT BI
T 0
                        512*                                        ;  AS IS INDICATED IN SYNERTEK LITERATURE
                        513* ,-------------------------------------
            00000000    514* CM_TDHI    EQU     0                   ;XMIT DISABLED RTS BAR HI
            00000004    515* CM_TELO    EQU     $4                  ;XMIT ENABLED RTS BAR LO
            00000008    516* CM_TDLO    EQU     $8                  ;XMIT DISABLED RTS BAR LO
            0000000C    517* CM_TDBRK   EQU     $C                  ;XMIT DISABLED --XMIT BREAK
                        518* ,
                        519* ;                   SOME USEFUL MACRO COMMANDS
                        520* ;                   FOR THE COMMAND REGISTER
            00000002    521* TURNOFF    EQU     CM_IRQD
            00000004    522* XMITENB    EQU     CM_TELO
            00000008    523* XMITDIS    EQU     CM_TDLO
            00000009    524* CMDRC      EQU     CM_DTRL+CM_TDLO     ;NO XMIT INT, RCV INT,ENAB DTR, NO PARITY
            00000005    525* CMDRWC     EQU     CM_DTRL+CM_TELO     ;SAME AS CMDRC XCEPT XMIT INTERRUPTS ENABLED
 ALSO
                        526* ;
                        527* ;-------------------------------------
                        528* ,
            000000F3    529* CLRD3D2    EQU     $F3                  ;CLEAR BITS D3 & D2 A MASK
                        530* ;
                        531* ;*****************************************************************
                        532* ;
                        533* ;    UART CONTROL REGISTER EQUATES
                        534* ;
                        535* ;           NOTE: Baud is lower 4 bits of control word--see BAUDCNV table below
                        536* ;
            00000080    537* CR_STPB    EQU     $80                 ;IF 0 THEN = 1 STOP BIT
                        538*                                        ;  IF SET AS INDICATED = 2 STOP BITS IF NO P
ARITY
                        539*                                        ;                    =1 STOP BIT IF 8 BIT
CHAR + PARITY
                        540*                                        ;                    =1.5 STOP BITS IF 5BI
T WORD NO PARITY
                        541* ;-------------------------------------
            00000000    542* CR_WRDL8    EQU     0                   ;8 BITS WORD LENGTH
            00000020    543* CR_WRDL7    EQU     $20                 ;7 BIT WORD LENGTH
            00000040    544* CR_WRDL6    EQU     $40                 ; 6 ETC
            00000060    545* CR_WRDL5    EQU     $60                 ; 5 ETC.
                        546* ;-------------------------------------
            00000000    547* CR_EXTCLK   EQU     0                   ;EXTERNAL RECEIVE CLOCK
            AAAAAA1A    548* CR_BRGEN   EQU     $1A                 ;BAUD RATE GEN FOR CLOCK
```

```
                        551* , UART CONTROL REGISTER CONSTANTS FOR UART SETUP
                        552* ,
00000016                553* CTLRC       EQU     CR_BDGLK+CR_WRDL8       ,1 STOP BIT,8BIT WORD LENGTH,BAUD RATE GENE
RATOR
                        554*
                        555* . ASCII Control characters for printer control .
                        556* ,
00000011                557* XON         EQU     $11                     ,CAN XMIT (CTL-Q)
00000013                558* XOFF        EQU     $13                     ,STOP XMIT (CTL-S)
00000003                559* ETX         EQU     $03                     ,READY FOR MORE? (CTL-C)
00000005                560* ENQ         EQU     $05                     ,READY FOR MORE? (CTL-E)
00000006                561* ACK         EQU     $06                     ,YES, I'M READY (CTL-F)
00000000                562* NULL        EQU     $00                     ,NULL CHARACTER-DO NOTHING
0000000D                563* CR          EQU     $0D                     ,CARRIAGE RETURN
0000000A                564* LF          EQU     $0A                     ,LINE FEED
                        565* ,
                        566* . Maximum Parameter values for Unitstatus Set table entry functions
                        567* ,
00000006                568* MAXBAUD      EQU     6                       ,FOR SET BAUD RATE
00000004                569* MAXPRTY      EQU     4                       ,FOR SET PARITY
00000001                570* MAXWRDS      EQU     1                       ,FOR SET WORD SIZE
00000001                571* MAXDTCM      EQU     1                       ,FOR SET DATACOM
00000009                572* MAXHNDS      EQU     9                       ,FOR SET HANDSHAKE TYPE
00000085                573* MAXWHI       EQU     133                     ,HI WATER WRITE MAX # CHARS
00000050                574* MAXWLO       EQU     80                      ;LO WATER WRITE MAX # CHARS
00000085                575* MAXRHI       EQU     133                     ,HI WATER READ #CHARS MAX
00000050                576* MAXRLO       EQU     80                      ,LO WATER READ #CHARS MAX
                        577* ,
                        578* , error codes (IORESULT)
                        579* ,
00000003                580* INVCMD       EQU     IOEioreq                ,invalid cmd-(invalid I/O request)
00000032                581* INVTBLID     EQU     IOEtblid                ,invalid table id
00000036                582* INVPRM       EQU     IOEuiopm                ,invalid parameter
00000038                583* INVFNC       EQU     IOEfncod                ,invalid function code
                        584* ,
                        585* . Miscellaneous definitions
                        586* .
00000001                587* TRUE        EQU     1                       ; Pascal true boolean value
00000001                588* ON          EQU     1                       ,LISTING CONTROL - START LISTING
00000000                589* OFF         EQU     0                       ,LISTING CONTROL - STOP LISTING
000000F0                590* HILOMSK     EQU     $F0                     ,MASK OFF WATER MARKS -THRO THEM AWAY
```

```
592* ,
593* , UNIT I/O PARAMETER PASSING DEFINITION
594* ,
595* ,   COMMAND       UNIT   ADDR   COUNT   BLOCK   MODE     IORESULT   BUSY
596* , 0 - INSTALL     D0.W                                   D7.W
597* , 1 - READ        D0.W   D1.L   D2.W                     D7.W
598* ; 2 - WRITE       D0.W   D1.L   D2.W                     D7.W
599* , 3 - CLEAR       D0.W                                   D7.W
600* ; 4 - BUSY        D0.W                                   D7.W       D0.W
601* , 5 - STATUS      D0.W   D1.L   D2.W <--FUNCTION CODE    D7.W
602* ; 6 - UNMOUNT     D0.W                                   D7.W
603* ,
604* , ALL REGISTER VALUES ON ENTRY ARE SAVED AND RESTORED EXCEPT D0 & D7
605* ; INTERNAL REGISTER USEAGE :
606* ,
607* ,      D0      =       temp reg
608* ;      D1      =       temp reg
609* ,      D2      =       user's count
610* ,      D3      =       character to or from buffer
611* ;      D4      =       unit number
612* ,      D5      =       Mode flag 4 in DC ctl int rtns - VIA line bit 0
613* ;      D6      =       save of SR
614* ,      A0      =       temp reg
615* ,      A1      =       temp reg
616* ,      A2      =       temp reg
617* ,      A3      =       Parameter block address (user's data)
618* ,      A4      =       buffer address in SetupWB and SetupRB
619* ;      A5      =       UART base address
620* ,      A6      =       Pointer to port's data area
621* ;
```

```
                        623*                GLOBAL      COMDRV
                        624* ,
                        625* , DATACOM DRIVER
                        626* ,
0000                    627* COMDRV
0000  601E              628*                BRA.S       COM001                  ,JUMP AROUND HEADER
0002  00                629*                DATA.B      0                       ;DEVICE NOT BLOCKED
0003  1F                630*                DATA.B      31                      ;VALID CMDS - ALL VALID
0004  53 01 0E 00       631*                DATA.B      83,01,14,00             ;DATE
0008  17                632*                DATA.B      hmlen                   ,HEADER MSG LENGTH
0009  4441544143204452  633* xxx010         DATA.B      'DATACOMM driver (v 5.0)' ;HEADER MSG
0011  2856657273696F6E
0019  20352E3029
      00000017          634* hmlen          EQU         *-xxx010
                        635* ,
0020  7E03              636* COM001         MOVEQ       #INVCMD,D7              ;assume invalid command
0022  0C44  0006        637*                CMPI.W      #UNMCMD,D4              ;VALID COMMAND?
0026  622C              638*                BHI.S       PRNDERR                 ;NO
0028  48E7  7EFE        639*                MOVEM.L     D1-D6/A0-A6,-(SP)       ;SAVE REGISTERS
002C  4287              640*                CLR.L       D7                      ;CLEAR IORESULT
002E  2641              641*                MOVEA.L     D1,A3                   ;ADDRESS OF USERS BUFFER
0030  4286              642*                CLR.L       D6                      ,Clear save of SR register
0032  4DFA  09FA+        643*                LEA         PORT0Data, A6           ; assume talking to Port 0
0036  B07A  0EB0+        644*                CMP.W       Unit(PO, D0             ,is it Port 0?
003A  6704              645*                BEQ.S       COMisP0                 ;yes
003C  4DFA  0C4C+        646*                LEA         PORT1Data, A6           ,NO, talking to port 1
                        647* ,
0040  C144              648* COMisP0        EXG         D0, D4                  ;save unit number
0042  43FA  0012+        649*                LEA         COMTBL,A1               ;TURN THE COMMAND INTO A
0046  E348              650*                LSL.W       #1,D0                   ,INDEX TO THE FUNCTION
0048  3031  0000        651*                MOVE.W      0(A1,D0.W),D0
004C  4EB1  0000        652*                JSR         0(A1,D0.W)              ;DO FUNCTION
0050  4CDF  7F7E        653*                MOVEM.L     (SP)+,D1-D6/A0-A6       ;Restore registers
0054  4E75              654* PRNDERR        RTS
                        655* ;
                        656* ; THE PRINTER DRIVER JUMP TABLE
                        657* ;
0058  000E              658* COMTBL         DATA.W      COMINST-COMTBL          ,UNITINSTALL
005A  0164              659*                DATA.W      COMRD-COMTBL            ;UNITREAD
005C  0260              660*                DATA.W      COMWR-COMTBL            ;UNITWRITE
005E  05FE              661*                DATA.W      COMCLR-COMTBL           ;UNITCLEAR
0060  066E              662*                DATA.W      COMBSY-COMTBL           ,UNITBUSY
0062  06B0              663*                DATA.W      COMST-COMTBL            ,UNITSTATUS
0064  047C              664*                DATA.W      COMUNMT-COMTBL          ,UNITUNMOUNT
```

```
                        666* ;
                        667* ; COMINST - UNITINSTALL ==> SETUP THE DEFAULT BUFFER CONTROL FEATURES
                        668* ;   Assumes that a spurrious DataCom Control interrupt is benign and will
                        669* ;   be handled by the DataCom Control interrupt service routine correctly.
                        670* ;
                        671* ; save unit number and toggle common flag
                        672* ;
0064 4160               673* COMINST    BSR.S     SaveUnit
0066 612E               674*           BSR.S     DISINTS                 ;DISABLE DATACOM INTERRUPTS
                        675*
                        676* ; init buffer control table
                        677* ;
0068 41EE 006A          678*           LEA       BFRCTL(A6), A0          ;beginning of table
006C 43EE 0000          679*           LEA       DEFBWRT(A6), A1         ;beginning of default table
0070 7004               680*           MOVEQ     #DEFBCTLN-1, D0         ;number of words in table
                        681*
0072 30D9               682* CINbufctl  MOVE.W    (A1)+, (A0)+            ;move from default to real
0074 51C8 FFFC          683*           DBF       D0, CINbufctl           ;table is even number of words
                        684* ;
                        685* ; Initialise UART from constants and Printer Control Table & initialise VIA
                        686* ;
0078 13FC 0080 0063     687*           MOVE.B    #IODDRA,DDRA.L          ;INITIALIZE DATA DIRECTION REG FOR PORT A
007E 0F67
0080 6162               688*           BSR.S     SETUART
                        689* ;
                        690* ; Initialise READ, WRITE AND CONTROL BUFFER CONTROL TABLES
                        691* ;
0082 6100 00CE          692*           BSR       INIWRBF                 ;init write buffer
0086 6100 00EE          693*           BSR       INIRDBF                 ;init read buffer
008A 6100 0118          694*           BSR       INITCTLB                ;init control buffer
                        695* ;
                        696* ; Setup interrupt vectors
                        697* ;
008E 6100 0092          698*           BSR       SETVECS
                        699* ;
                        700* ; If saved SR then restore it
                        701* ;
0092 6128               702*           BSR.S     ENBINTS
0094 4E75               703*           RTS
```

```
                        705* ;
                        706* ; DISINTS - disable interrupts for Port selected.  If Port 0 then disable up to
                        707* ,          level 4.  If Port 1 selected than disable up to level 2.
                        708* ;
                        709* ,      Entry : D6 = saved SR if not zero
                        710* ;              D4 = unit number
                        711* ;      Exit  : D6 = saved SR or zero
                        712* ;
0096  323C  0400        713* DISINTS     MOVE.W     #INT4, D1               ;assume Port 0, level 4 int
009A  B87A  0E4C+       714*            CMP.W      UnitP0, D4              ;is it Port 0?
009E  6704              715*            BEQ.S      DITisP0                 ;yes
00A0  323C  0200        716*            MOVE.W     #INT2, D1               ;no, use Port 1 level 2 int
00A4  40C0              717* DITisP0    MOVE.W     SR,D0                   ;get current status register
00A6  0240  0700        718*            ANDI.W     #INTMSK,D0              ;GET ONLY INTERRUPT LEVELS
00AA  B041              719*            CMP.W      D1, D0                  ;is current ( current Port's level
00AC  640C              720*            BCC.S      DITexit                 ;no, exit
                        721* ;
                        722* ;    NOW set up disable with minimum disturbance of upper level
                        723* ,              status bits --- this too wont work if user and
                        724* ,              supervisor space are both utilised.
                        725* ;
00AE  40C6              726*            MOVE.W     SR, D6                  ;save current SR
00B0  40C0              727*            MOVE.W     SR, D0                  ;get current status register for change
00B2  0240  A000        728*            ANDI.W     #UPRMSK,D0              ;KEEP ONLY UPPER BITS
00B6  8041              729*            OR.W       D1, D0                  ;disable current Port's level
00B8  46C0              730*            MOVE.W     D0,SR                   ;turn off the ints in the SR
00BA  4E75              731* DITEXIT    RTS
                        732* ;
                        733* ; ENBINTS - Restore saved SR if saved it
                        734* ,      Entry : D6 = saved SR if not zero
                        735* ;      Exit  : D6 = if D6 was not zero then SR <- D6 and D6 <- 0
                        736* ;                  otherwise SR remains untouched and D6 stays 0
                        737* ;
00BC  4A46              738* ENBINTS     TST.W      D6                      ;Does D6 have a saved SR
00BE  6704              739*            BEQ.S      EITEXIT                 ;DIDN'T SAVE SO EXIT
00C0  46C6              740*            MOVE.W     D6,SR                   ;restore SR
00C2  4286              741*            CLR.L      D6                      ;always leave D6 = to zero
00C4  4E75              742* EITEXIT    RTS
```

```
                        744* ;
                        745* ; SaveUnit - determine if this is Port 0 or Port 1 and save unit number
                        746* ,            also initialise A6 to address of port's data area
                        747* ;
                        748* ;       Entry : D4 = unit number
                        749* ,       Exit  : A6 = address of port's data area
                        750* ;
00C6  43FA  0E20+       751* SaveUnit    LEA       Unit#0, A1              ;assume is Port 0
00CA  4DFA  0962+       752*             LEA       Port0Data, A6
                        753*
00CE  41FA  0E16+       754*             LEA       CMNFLGS, A0             ;it portflg flag was
00D2  0850  0000        755*             BCHG      #PORTFLG, (A0)          ; zero then is port 0
00D6  6708              756*             BOFF.S    SVUisF0                 ;else it is now port 1
                        757*
00D8  43FA  0E10+       758*             LEA       Unit#1, A1              ;Port 1 addresses
00DC  4DFA  0BAC+       759*             LEA       Port1Data, A6
                        760*
00E0  3284              761* SVUisF0     MOVE.W    D4, (A1)                ;save unit number
00E2  4E75              762*             RTS
```

```
                        764* ;
                        765* ; SETUART - Initialize UART from constants and Buffer Control Table
                        766* ;
                        767* , Get UART Register Base address
                        768* ,
00E4  612A              769* SETUART     BSR.S       GETBASE                 ;RETURNS BASE IN A0
                        770* ,
                        771* , Setup UART's Control register - index = 7  from Base
                        772* ,
00E6  7010              773*            MOVEQ       #CTLRC,D0               ,1 STOP BIT,BAUD RATE GEN
00E8  122E  000D        774*            MOVE.B      BF_WRDS(A6), D1         ,ADD WORD SIZE-7 OR 8 BITS
00EC  EB09              775*            LSL.B       #5,D1                   ,MOVE INTO HI ORDER BITS
00EE  8001              776*            OR.B        D1,D0                   ,00=8 BITS,01=7 BITS
00F0  802E  000E        777*            OR.B        BF_RDBD(A6), D0         ,ADD BAUD RATE FROM TABLE
00F4  1B40  0007        778*            MOVE.B      D0,CTLREGI(A5)          ,PUT IN CONTROL REGISTER
                        779* ,
                        780* , Setup UART's Command register - index = 5 from Base
                        781* , make transmit buffer empty interrupt enabled - when occurs int rtn will
                        782* ,  turn off if buffers are empty.
                        783* ;
00F8  7005              784*            MOVEQ       #CMDRWC, D0             ;CMD CONSTANTS xmit int enabled
00FA  122E  000C        785*            MOVE.B      BF_PART(A6), D1         ;GET TABLE PARITY
00FE  EB07              786*            LSL.B       #5,D1                   ,PUT IN CORRECT BIT POSITION
0100  8001              787*            OR.B        D1,D0
0102  1B40  0005        788*            MOVE.B      D0,CMDREGI(A5)          ,PUT IN COMMAND REGISTER
                        789* ,
                        790* , Read the Data Port and Status Register to clear all Status flags
                        791* ,
0106  102D  0001        792*            MOVE.B      DATAREG(A5),D0          ,DATA PORT AT INDEX = 1
010A  102D  0003        793*            MOVE.B      STATRI(A5),D0           ;STATUS REG AT INDEX = 3
010E  4E75              794*            RTS
```

```
                                796* ;
                                797* ; GETBASE - Get address of UART's register Base address in memory
                                798* ;          Entry - D4 = unit number
                                799* ;          EXIT  - (A5) = Base address
                                800* ;
0110 4BF9 0003 0F20             801* GETBASE    LEA       UARTDC0.L, A5         ;ASSUME USING DATACOM 0
0116 B87A 0DD0+                 802*            CMP.W     UnitP0, D4            ;is it Port 0?
011A 6704                       803*            BEQ.S     GBSisP0               ;yes
011C DAFC 0020                  804*            ADDA.W    #DC1OFF, A5           ;No. BASE = OFFSET+UART DC0 BASE ADDR
0120 4E75                       805* GBSisP0    RTS
                                806* ;
                                807* ; SETVECS - Put interrupt routine's entry addresses into the interrupt vectors
                                808* ;          If Port 0 put in DC 0 int rtn address in Vector 4
                                809* ;          otherwise assume is port 1
                                810* ;          Saves old level 1 interrupt vector if it is not = to this driver's
                                811* ;             interrupt routines address.
                                812* ;
                                813* ;          Entry   D4 = unit number
                                814* ;                  interrupts disabled to level for current Port
                                815* ;
0122 41FA 049C+                 816* SETVECS    LEA       DCTLINT,A0            ;PUT DATA COM CONTROL
0126 2278 0064                  817*            MOVEA.L   VEC1.W, A1            ;Get old vector
012A 21C8 0044                  818*            MOVE.L    A0,VEC1.W             ;INT ROUTINE IN VEC 1
012E B3C8                       819*            CMPA.L    A0, A1                ;should save old vector
0130 6704                       820*            BEQ.S     SVCsame               ;no. they're the same
0132 41FA 0DB8+                 821*            LEA       SaveLvl1, A0          ;yes save in common area
0136 2089                       822*            MOVE.L    A1, (A0)
                                823*
0138 B87A 0DAE+                 824* SVCsame    CMP.W     UnitP0, D4            ;is it Port 0?
013C 660A                       825*            BNE.S     SVCdoP1               ;no, do level 2 for Port 1
                                826*
013E 41FA 025C+                 827*            LEA       DC0INT,A0             ;ADDR OF DC0 entry point to XMIT/RCV INT RO
UTINE
0142 21C8 0070                  828*            MOVE.L    A0,VEC4.W             ;put it in VEC 4
0146 600A                       829*            BRA.S     SVCexit
                                830*
0148 41FA 0260+                 831* SVCdoP1    LEA       DC1INT,A0             ;ADDR OF DC1 entry point to XMIT/RCV INT RO
UTINE
014C 21C8 0068                  832*            MOVE.L    A0,VEC2.W             ;put it in VEC 2
                                833*
0150 4E75                       834* SVCexit    RTS
```

```
                        836* ;
                        837* ; INIWRBF - Initialize Write Buffer variables to EMPTY Buffer also ENQ, BUSY and
                        838* ;          SENDLF are cleared to false.  Use default buffer
                        839* ;
0152  41EE  0014        840* INIWRBF    LEA       WRTCTL(A6),A0           ;WRITE BUFFER CONTROL TABLE
0156  4258               841*           CLR.W     (A0)+                   ;RESET ALL FLAG 1
0158  4218               842*           CLR.B     (A0)+                   ;RESET ALL FLAG 2 except
015A  08D0  0004         843*           BSET      #AULF_W2,(A0)           ; DO AUTO LINE FEED and *kb 1/5/83*
015E  08D8  0001         844*           BSET      #EMPT_W2,(A0)+          ; BUFFER IS EMPTY
0162  43EE  015C         845*           LEA       WRTBUF(A6), A1          ;WRITE BUFFER
0166  2DC9               846*           MOVE.L    A1,(A6)+                ;FILL POINTER (USED TO FILL CHARACTERS IN)
0168  2DC9               847*           MOVE.L    A1,(A0)+                ;EMPTY POINTER (USED TO EMPTY CHARACTERS OU
T:
016A  2DC7               848*           MOVE.L    A1,(A0)+                ;Save buffer address
016C  30FC  0100         849*           MOVE.W    #WBFLEN,(A0)+           ;MAXIMUM SIZE OF BUFFER
0170  30FC  0100         850*           MOVE.W    #WBFLEN,(A0)+           ;NUMBER OF LOCATIONS AVAILABLE TO FILL
0174  4E75               851*           RTS
                        852* ;
                        853* ; INIRDBF - Initialize READ Buffer variables to EMPTY Buffer also ENQ, BUSY and
                        854* ;          SENDLF are cleared to false.  Use default buffer
                        855* ;
0176  41EE  0030        856* INIRDBF    LEA       RDCTL(A6), A0           ;READ BUFFER CONTROL TABLE
017A  4258               857*           CLR.W     (A0)+                   ;RESET ALL FLAG 1
017C  4218               858*           CLR.B     (A0)+                   ;RESET ALL FLAG 2 except,
017E  08D8  0001         859*           BSET      #EMPT_W2,(A0)+          ;   BUFFER IS EMPTY
0182  43EE  005C         860*           LEA       RDBUF(A6),A1            ;READ BUFFER
0186  2DC9               861*           MOVE.L    A1,(A0)+                ;FILL POINTER (USED TO FILL CHARACTERS IN)
0188  2DC9               862*           MOVE.L    A1,(A0)+                ;EMPTY POINTER (USED TO EMPTY CHARACTERS OU
T:
018A  2DC7               863*           MOVE.L    A1,(A0)+                ;Save buffer address
018C  30FC  0100         864*           MOVE.W    #RBFLEN,(A0)+           ;MAXIMUM SIZE OF BUFFER
0190  30FC  0100         865*           MOVE.W    #RBFLEN,(A0)+           ;NUMBER OF LOCATIONS AVAILABLE TO FILL
0194  4298               866*           CLR.L     (A0)+                   ;Clear alternate buffer address
0196  4258               867*           CLR.W     (A0)+                   ;Clear alternate buffer length
0198  30FC  0085         868*           MOVE.W    #MAXRHI,(A0)+           ;NUMBER OF CHARACTERS FOR HIGH WATER MARK
019C  30FC  0050         869*           MOVE.W    #MAXRLO,(A0)+           ;NUMBER OF CHARACTERS FOR LOW WATER MARK
01A0  4258               870*           CLR.W     (A0)+                   ;CLEAR ENQ COUNT
01A2  4E75               871*           RTS
                        872* ;
                        873* ; INITCTLB  - Initialize the control character buffer to empty
                        874* ;
01A4  41EE  0058        875* INITCTLB    LEA       CTLBUF(A6), A0          ;CONTROL CHARACTER BUFFER
01A8  43EE  004E         876*           LEA       CB_FRONT(A6), A1        ;CTL CHAR BUF TABLE ADDRESS
01AC  22C8               877*           MOVE.L    A0, (A1)+               ;set front and
01AE  22C8               878*           MOVE.L    A0, (A1)+               ;rear pointers to begin of buffer
01B0  4251               879*           CLR.W     (A1)                    ;clear all flags except
01B2  08E9  0001  0001   880*           BSET      #EMPT_CB, 1(A1)         ;buffer empty
01B8  4E75               881*           RTS
```

```
                        883* , COMRD - UNITREAD   READ FROM THE DATACOM BUFFER
                        884* ,
                        885* ,           INPUTS....D2 COUNT OF CHARACTERS THE USER WANTS TO READ
                        886* ;                    D4 unit number
                        887* ,                    A3 ADDRESS OF USER'S BUFFER
                        888* ,                    A6 Address of ports data area
                        889* ,
                        890* ,  NOTES  for reading, interrupts will occur when the input buffer is full -no
                        891* ,        priming is necessary as is with writing. Also if full duplex activities
                        892* ,        then a read and write interrupt may be the same interrupt -have to check
                        893* ,        status flags of UART
                        894* ; First see if user's count is exhausted if not attempt a read
                        895* ,
01BA  4A42              896* COMRD     TST.W    D2
01BC  6736              897*           BEQ.S    COMREX                  ,COMREX GENERAL EXIT ROUTINE
                        898*
                        899* , Check if the user has disabled output - Buffer to User
                        900* ,
01BE  082E  0004  0031  901* REREAD    BTST     #OUTE_R1.RB_FLG1+1(A6)  ,IS BUFFER TO USER TRANSFER ENABLED?
01C4  6704              902*           BEQ.S    CKRdErr                 , YES, check for input error on UART
01C6  7E3D              903*           MOVEQ    #IOEordsbl,D7           ,no, tell user can't =) ERROR
01C8  601A              904*           BRA.S    COMREX
                        905*
                        906* , Check for a UART error
                        907* ;
01CA  08AE  0001  0031  908* CKRdErr   BCLR     #Err_R1, RB_FLG1+1(A6)  ,Have a read error
01D0  6704              909*           BCFF.S   CKPORT                  ,no, see if have data
01D2  7E43              910*           MOVEQ    #IOEuarter,D7           ,yes, tell user and exit
01D4  601E              911*           BRA.S    COMREX
                        912*
                        913* , if there is any data in the buffer, give it to user. If there is no data and
                        914* ,  the user has disabled the inbound read, remind him. However if there is no data
                        915* ;  put him in a loop waiting for data.
                        916* ,
01D6  082E  0001  0033  917* CKPORT    BTST     #EMPT_R2,RB_FLG2+1(A6)  ,BUFFER IS EMPTY?
01DC  670C              918*           BOFF.S   READONE                 , NO, GO READ A CHARACTER
01DE  082E  0005  0031  919*           BTST     #INPE_R1,RB_FLG1+1(A6)  , yes, INPUT ENABLED?
01E4  67D8              920*           BOFF.S   REREAD                  , YES, wait for a char
01E6  7E3C              921*           MOVEQ    #IOEirdsbl,D7           ,no, tell user input is disabled
01E8  600A              922*           BRA.S    COMREX
                        923*
                        924* , get user his characters and manage buffer
                        925* ,
01EA  610A              926* READONE   BSR.S    UGETCHR                 ,GET THE CHARACTER FOR THE USER FROM THE BUF
FER
01EC  6506              927*           BCS.S    COMREX                  ,exit if error, D7 has error code
                        928*
                        929* ; Put character in user's buffer and update
                        930* ;
01EE  16C3              931*           MOVE.B   D3,(A3)+                ,update buffer pointer
01F0  5342              932*           SUBQ.W   #1,D2                   ,subtract one from user count
01F2  60C6              933*           BRA.S    COMRD                   ,GETSMOA IF AVAILABLE
                        934* ,
01F4  4E75              935* COMREX    RTS
```

```
                        937* ; UGETCHR  --- User level get character routine, gets the character from the read buffer.
                        938* ;
                        939* ;       Entry     A6  = pointer to ports data area
                        940* ;                 D2  = user count
                        941* ;                 D4  = unit number
                        942* ;                 buffer is NOT empty
                        943* ;       Exit      D3  = character if one gotten
                        944* ;                 (C) = Error, D7 has error code
                        945* ;                (NC) = got a character no error
                        946* ;
01F6  6100  FE9E        947* UGETCHR    BSR       DISINTS                 ;disable interupts
                        948*
01FA  206E  0038        949*           MOVE.L    RB_EMPTY(A6), A0        ;A0 =/ EMPTYING POSITION OF RD BUFFER
01FE  1618              950*           MOVE.B    (A0)+,D3                ;Get chars
0200  2D48  0038        951*           MOVE.L    A0, RB_EMPTY(A6)        ;Save the new Front pointer in rb_empty
                        952* ;
                        953* ; Update buffer variables
                        954* ;
0204  226E  003C        955*           MOVE.L    RB_BADR(A6), A1         ;A1 = ADDRESS OF BUFFER BEGIN
0208  D2EE  0040        956*           ADDA.W    RB_SIZE(A6), A1         ;A1 = ADDRESS OF END OF BUFFER
020C  B1C9              957*           CMPA.L    A1, A0                  ;Is Front pointing beyond buffer?
020E  6306              958*           BLS.S     UGCnowrp                ;No, don't do wrap around
                        959* ;                                           ;Yes, set front = addr 1st byte of buffer
0210  2D6E  003C  0038  960*           MOVE.L    RB_BADR(A6), RB_EMPTY(A6) ;Save the new Front pointer in rb_empty
                        961* ;
0216  526E  0042        962* UGCnowrp   ADDQ.W    #1, RB_FREE(A6)         ;SINCE WE GOT CHAR, ONE MORE FREE SPACE
                        963* ;
                        964* ; see if buffer is empty
                        965* ;
021A  322E  0042        966*           MOVE.W    RB_FREE(A6), D1
021E  B26E  0040        967*           CMP.W     RB_SIZE(A6), D1         ; # OF FREE LOCATIONS - BUFFER SIZE
0222  650A              968*           BCS.S     UGCnotmt                ; not empty if free < size
0224  6212              969*           BHI.S     HELPRD                  ; HELPRD IS SERIOUS ERROR (free > size)
                        970* ;
0226  08EE  0001  0033  971*           BSET      #EMPT_R2, RB_FLG2+1(A6) ; BUFFER empty
022C  615C              972*           BSR.S     RChkAltBf               ; see if should switch to an Alternate buffe
                        973* ;
                        974* ; Do protocol control, see if can turn off Read Busy
                        975* ;
022E  6114              976* UGCnotmt   BSR.S     ChkProto                ;check protocol
0230  6100  FE8A        977*           BSR       ENBINTS                 ;enable interrupts
0234  4280              978*           CLR.L     D0                      ; clear carry
0236  4E75              979*           RTS
                        980* ;
0238  6100  FE82        981* HELPRD     BSR       ENBINTS                 ; SERIOUS BUMMER BUG
023C  7E40              982*           MOVEQ     #IOEdsserr,D7           ;SIZING ERROR
023E  44FC  0001        983*           MOVE.W    #1,CCR                  ;SET CARRY
0242  4E75              984*           RTS
```

```
                        986* ;
                        987* ;  ChkProto  - checks low water mark for reading to see if should turn off read busy
                        988* ;  GoUnbusy - entry point to turn off busy state on receives
                        989* ;
                        990* ;          NOTES: This routine assumes that interrupts are disabled prior
                        991* ;                 to its being invoked
0244                    992* ChkProto
                        993* ;
                        994* ; check if input disabled. Cannot turn off busy if is disabled.
                        995* ;
0244 082E 0005 0031     996*            BTST        #INPE_R1, RB_FLG1+1(A6)
024A 663C               997*            BNE.S       CPRexit                 ;input disabled exit
                        998* ;
                        999* ; if (protocols enabled) and (NOT Line type) then check if busy
                       1000* ;
024C 082E 0000 0012    1001*            BTST        #PROT_P2, RF_PROF(A6)   ;protocol enabled?
0252 6734              1002*            BEQ.S       CPRexit                 ;no, exit
0254 082E 0000 0013    1003*            BTST        #LINE, RF_PROF+1(A6)    ;Line type
025A 662C              1004*            BNE.S       CPRexit                 ;yes, exit
                       1005* ;
                       1006* ; if busy then check if buffer at or below low water mark
                       1007* ;
025C 082E 0000 0031    1008*            BTST        #BUSY_R1, RB_FLG1+1(A6)
0262 6724              1009*            BEQ.S       CPRexit                 ;not busy exit
                       1010* ;
                       1011* ; if buffer at or below low water mark then turn off busy
                       1012* ;
0264 322E 0040         1013*            MOVE.W      RB_SIZE(A6), D1         ;BUFFER SIZE (ADDRESS CF)
0268 926E 0042         1014*            SUB.W       RB_FREE(A6), D1         ;D1 = number of chars in buffer
026C B26E 004C         1015*            CMP.W       RB_LOWA(A6), D1         ; at or below low water mark?
0270 6216              1016*            BHI.S       CPRexit                 ; No. exit
                       1017* ;
                       1018* ; is at or below when busy so turn off busy
                       1019* ;
0272 08AE 0000 0031    1020* GoUnbusy   BCLR        #BUSY_R1, RB_FLG1+1(A6) ;clear busy state
0278 7011              1021*            MOVEQ       #XON, D0                ;assume XON/XOFF protocol
027A 082E 0001 0013    1022*            BTST        #XONXOFF, RF_PROF+1(A6) ;send byte to other side saying not busy
0280 6602              1023*            BNE.S       CPRxon                  ;send XON
0282 7006              1024*            MOVEQ       #ACK, D0                ;either ETX/ACK or ENQ/ACK so send ACK
0284 6100 00EC         1025* CPRxon     BSR         PutCtl                  ;send the control char
0288 4E75              1026* CPRexit    RTS
```

```
                        1026* ,
                        1027* , RChkAltBf - check if alternate buffer switch on read buffer
                        1028* ,       Rcv input is automatically disabled when user calls unitstatus
                        1029* ,       switch buffers.
                        1030* ,
                        1031* ,       Entry    A0 = address of port's datat area
                        1032* ,                interrupts disabled
                        1033* ,
02BA  082E  0001 0033   1034* RChkAltBf   BTST       #EMPT_R2, RB_FLG2+1(A6)  ,is buffer empty?
02BE  6722              1035*             BEQ.S      rCABexit                 ,no, can't switch
02C0  08AE  0002 0031   1036*             BCLR       #ALTBF_R1, RB_FLG1+1(A6) ,is an alternate buffer available?
02C6  670A              1037*             BEQ.S      rCABexit                 ,no, nothing to switch
                        1038* ,
                        1039* , Switch buffers by making the Alternate buffer the main buffer
                        1040* ,
02CA  206E  0044        1041*             MOVE.L     RB_ABADR(A6), A0         ,get new buffer address
02CE  302E  0048        1042*             MOVE.W     RB_ABSIZE(A6), D0        ,and length
02D2  6100  0750        1043*             BSR        SetupRB                  ,switch buffer in table
                        1044* ,
                        1045* , if user is NOT controlling the input disable bit then enable RCV input
                        1046* ,
02D6  41EE  0031        1047*             LEA        RB_FLG1+1(A6), A0        ,EnbRcvIn needs A0 -> flag byte
02DA  0810  0000        1048*             BTST       #INPC_R1, (A0)           ,is user controlling input disable?
02DE  6604              1049*             BNE.S      rCABexit                 ,user is controlling, exit
02E0  6100  0670        1050*             BSR        EnbRcvIn                 ,enable recevie input
                        1051* ,
02E4  4E75              1052* rCABexit    RTS
```

```
                        1056* , COMWR - UNITWRITE
                        1057* ,
                        1058* ,           INPUTS  ..  D2 COUNT OF CHARACTERS THE USER WANTS TO WRITE
                        1059* ,                       D4 unit number
                        1060* ,                       A3 ADDRESS OF USER'S CHARACTERS
                        1061* ,                       A4 Address of ports data area
                        1062* ,
                        1063* , NOTE  For writing, the UART has to be primed to interrupting when the xmit buffer
                        1064* ,       is empty by enabling the xmit interrupt  If no xmissions then of course its empty
                        1065* ,       and it interrupts forever  Hence trickery only when sending first of a stream
                        1066* ,       (starting interrupts) and last of a stream (stopping the little dears)
                        1067* ,
02E6  4A42              1068* COMWR     TST.W     D2                      ,IS USER COUNT DONE?
02E8  673E              1069*           BEQ.S     COMWEX                  ,YES
                        1070*
                        1071* , if input to buffer disabled input
                        1072* ,
02EA  083E 0005 0015    1073* REWRITE   BTST      #INPE_W1, WB_FLG1+1(A6) ,IS USER TO BUFFER TRANSFER ENABLED?(INBOUND
WRITE)
02EE  6704              1074*           BEQ.S     CKbuferr                ,YES, chk if buffer size err found in xmit i
nterrupt
02F0  7E3E              1075*           MOVEQ     #IOEiwdsbl,D7           ,input disabled give error
02F2  6032              1076*           BRA.S     WRPROB                  ,exit
                        1077*
                        1078* , check write error flag for error during xmit interrupt
                        1079* ,
02F4  0C2E 0001 0012    1080* CKbuferr  BCLR      #ERR_W1, WB_FLG1+1(A6)  ,Error?
02FA  6704              1081*           BEQ.S     CKWRTP                  ,No, chk if buffer is full
02FC  7E41              1082*           MOVEQ     #IOEwsserr,D7           ;SIZING ERROR with write buffer
02FE  602A              1083*           BRA.S     WRPROB                  ,exit
                        1084*
                        1085* , Check if Buffer is full  If is and output is NOT disabled then spin wheels
                        1086* ,
0300  082E 0000 0017    1087* CKWRTP    BTST      #FULL_W2, WB_FLG2+1(A6) ,Buffer full?
0306  670C              1088*           BEQ.S     WRTONE                  ,NO, GO WRITE A CHAR TO THE BUFFER
                        1089*
0308  083E 0004 0015    1090*           BTST      #OUTE_W1, WB_FLG1+1(A6) , Yes, OUTPUT IS AT ALL ENABLED?
030E  67D8              1091*           BEQ.S     REWRITE                 , YES, spin wheels while buffer emptys
0310  7E3F              1092*           MOVEQ     #IOEowdsbl,D7           ,can't send tell user ERROR
0312  6012              1093*           BRA.S     WRPROB                  ,exit
                        1094*
                        1095* , Buffer not full so put user characters or LF into write buffer
                        1096* ,
0314  760A              1097* WRTONE    MOVEQ     #LF, D3                 ,assume just sent an CR so must send a LF
0316  0BAE 0003 0017    1098*           BCLR      #SNDLF_W1, WB_FLG2+1(A6)  ,should send an Line Feed char?
031C  6604              1099*           BNE.S     WRTanLF                 ,yes
031E  161B              1100*           MOVE.B    (A3)+,D3                ,no, then get 1 user char
0320  5342              1101*           SUBQ.W    #1,D2                   ,subtract 1 from user's count
0322  6104              1102* WRTanLF   BSR.S     UPUTCHR                 ,PUT THE USER'S CHARACTER INTO THE WRITE BUF
FER
0324  60BE              1103*           BRA.S     COMWR
0326                    1104* COMWEX
0326  4E75              1105* WRPROB    RTS
```

```
                        1107* ; UPUTCHR  --- User level put character routine, puts the character into the write buffer.
                        1108* ,
                        1109* ;       Entry : (D3) = character to put in write buffer
                        1110* ;               Buffer is NOT full
                        1111* ;
62FA  6100  FD9A        1112* UPUTCHR     BSR       DISINTS                 ,disable interrupts
                        1113*
62FE  206E  0018        1114*             MOVE.L    WB_FILLP(A6), A0        ,A0 =) FILLING POSITION OF WRITE BUFFER
0302  10C3              1115*             MOVE.B    D3, (A0)+               ;PUT char
0304  2D48  0018        1116*             MOVE.L    A0, WB_FILLP(A6)        ,Save the new Rear pointer in wb_fillp
                        1117* ;
                        1118* , Update buffer variables
                        1119* ;
0308  226E  0020        1120*             MOVE.L    WB_BADR(A6), A1         ,A1 = ADDRESS OF BUFFER BEGIN
030C  D2EE  0024        1121*             ADDA.W    WB_SIZE(A6), A1         ,A1 = ADDRESS OF END OF BUFFER
0310  B1C9              1122*             CMPA.L    A1, A0                  ;Is Rear pointing beyond buffer?
0312  6306              1123*             BLS.S     UPCnowrp                ;No, don't do wrap around
                        1124* ,                                             ;Yes, set front = addr 1st byte of buffer
0314  2D6E  0020  0018  1125*             MOVE.L    WB_BADR(A6), WB_FILLP(A6) ;Save the new Rear pointer in wb_fillp
                        1126* ;
031A  536E  0026        1127* UPCnowrp     SUBQ.W    #1, WB_FREE(A6)         ;SINCE WE TOOK CHAR, ONE LESS FREE SPACE
                        1128* ;
                        1129* ; see if buffer is full  (WB_FREE is an unsigned word)
                        1130* ,
031E  6606              1131*             BNE.S     UPCnotfl                ,not full, subtract sets or clears ZERO bit
0320  08EE  0000  0017  1132*             BSET      #FULL_W2, WB_FLG2+1(A6)  , BUFFER full
                        1133* ;
                        1134* , check if last char is CR.  If is see if should send an LF next time
                        1135* ,
0324  0C03  000D        1136* UPCnotfl     CMPI.B    #CR, D3
0328  6616              1137*             BNE.S     UPCnotCR                ;not a CR
032A  082E  0004  0017  1138*             BTST      #AULF_W2, WB_FLG2+1(A6) ;is it auto LF mode
0330  670E              1139*             BOFF.S    UPCnotCR                ,no, don't send an LF  *kb 1/5/82*
0332  3005              1140*             MOVE.W    D5, D0                  ,save mode flag  *kb 1/24/83*
0334  0240  000C        1141*             ANDI.W    #LFsprsflg, D0          ,if LF suppress flag set *kb 1/24/83*
0338  6606              1142*             BNE.S     UPCnotCR                ;then don't send a LF
033A  08EE  0003  0017  1143* UPCisaLF     BSET      #SNDLF_W2, WB_FLG2+1(A6) ,send LF only if D5=0 and AULF set
                        1144*
                        1145* ; show buffer not empty.  If was output char and and turn on xmit interrupts
                        1146* ;
033E  08AE  0001  0017  1147* UPCnotCR     BCLR      #EMPT_W2, WB_FLG2+1(A6) ,test and clear
0344  670A              1148*             BOFF.S    UPCison                 ;wasn't empty before
0346  082E  0004  0015  1149*             BTST      #OUTE_W1, WB_FLG1+1(A6) ;if output to user is disabled
034C  6602              1150*             BON.S     UPCison                 ;then don't start xmit int
                        1151*
                        1152* , interrupt will occur without sending a char
034E  6184              1153*             BSR.S     STRTXMIT                ;turn on interrupt
                        1154*
                        1155* ; enable interrupts and exit
                        1156* ;
0350  6000  FD44        1157* UPCison     BRA       ENBINTS
```

```
                    1159*  ,          NOTE:    it is assumed that these routines are protected from interrupts
                    1160*  ,
                    1161*  ; STRTXMIT - start xmit interrupt process by enabling UART to interrupt
                    1162*  ;            on transmit buffer empty.
                    1163*  ; STOPXMIT - stop xmit interrupt process by disabling UART to interrupt
                    1164*  ;            on transmit buffer empty.
                    1165*  ,     Entry : D4 = unit number
                    1166*  ;
0358                1167*  STRTXMIT
0358  7204          1168*            MOVEQ     #XMITENB,D1             ,ENABLE XMIT INT
035A  6002          1169*            BRA.S     SITGETB
035C                1170*  STOPXMIT
035C  7208          1171*            MOVEQ     #XMITDIS,D1             ,DISABLE XMIT INT
                    1172*  ,
035E  6100  FDB0    1173*  SITGETB   BSR       GETBASE                 ,GET UART BASE ADDRESS
0362  102D  0005    1174*            MOVE.B    CMDREGI(A5),D0          ,GET CURRENT CMD REG
0366  0200  00F3    1175*            ANDI.B    #CLRD3D2,D0             ,CLEAR BITS D3 & D2
036A  8001          1176*            OR.B      D1,D0                   ;DON'T CHANGE OTHER BITS
036C  1B40  0005    1177*            MOVE.B    D0,CMDREGI(A5)          ;SAVE CHANGED CMD REG
0370  4E75          1178*            RTS
```

```
                        1180* ;
                        1181* ; PutCtl - put a character in the control character buffer
                        1182* ;      Entry : (D0) = character to put in control char buffer
                        1183* ;                     interrupts disabled
                        1184* ;
0372  206E  0052        1185* PutCtl     MOVEA.L    CB_REAR(A6), A0          ;A0 = Rear pointer
0376  10C0              1186*            MOVE.B     D0, (A0)+                ;put char in buffer and inc ptr
0378  2D48  0052        1187*            MOVE.L     A0, CB_REAR(A6)          ;put Rear pointer in save loc
037C  08AE  0001  0057  1188*            BCLR       #EMPT_CB, CB_FLAGS+1(A6) ;show not empty
0382  6004              1189*            BRA.S      STRTXMIT                 ;make sure will send character
                        1190* ,
                        1191* ; GetCtl - get a character from the control character buffer
                        1192* ,      Exit  : (D3) = character to from control char buffer
                        1193* ;                     interrupts disabled
                        1194* , Assumption :  The control buffer should never get full.
                        1195* ,
0384  206E  004E        1196* GetCtl      MOVEA.L    CB_FRONT(A6), A0         ;A0 = Front pointer
0388  1618              1197*            MOVE.B     (A0)+, D3                ;get char from buffer and inc ptr
038A  2D48  004E        1198*            MOVE.L     A0, CB_FRONT(A6)         ;put Front pointer in save loc
                        1199*
038E  222E  0052        1200*            MOVE.L     CB_REAR(A6), D1          ;D1 = Rear pointer
0392  B288              1201*            CMP.L      A0, D1                   ;Front = Rear?
0394  6204              1202*            BHI.S      GCLexit                  ;no, still more chars in buffer
                        1203*                                                ;yes, buffer empty
0396  6100  FE0C        1204*            BSR        INITCTLB                 ;init control buffer to empty
                        1205*
039A  4E75              1206* GCLexit     RTS
```

```
                        1208* ;
                        1209* ; DCOMINT - DataCom interrupt routine for XMIT/RCV interrupts.
                        1210* ;
                        1211* ; CRITICAL: if an interrupt occurs, then both the receive buffer full and the xmit
                        1212* ;           buffer empty could be true simultaneously, so we must test both.
                        1213* ;               However, only once thru the test then rte
                        1214* ;           Currently the priority is reads then writes
                        1215*
                        1216* ; Entry for Port 0 interrupt
                        1217* ;
039C 48E7 FFFE          1218* DC0INT      MOVEM.L   D0-A6,-(SP)             ;SAVE ALL REGISTERS
03A0 4DFA 048C+         1219*             LEA       Port0Data, A6           ;Address of Port 0 data
03A4 383A 0842+         1220*             MOVE.W    Unit0, D4               ;Port 0 unit number
03A8 600C               1221*             BRA.S     DCIcomm
                        1222*
                        1223* ; Entry for Port 1 interrupt
                        1224* ;
03AA 48E7 FFFE          1225* DC1INT      MOVEM.L   D0-A6,-(SP)             ;SAVE ALL REGISTERS
03AE 4DFA 08DA+         1226*             LEA       Port1Data, A6           ;Address of Port 1 data
03B2 383A 0836+         1227*             MOVE.W    Unit1, D4               ;Port 1 unit number
                        1228*
                        1229* ; begin of Common port interrupt code
                        1230* ;
03B6 6100 FD56          1231* DCIcomm     BSR       GETBASE                 ;get UART base address
03BA 1E1D 0803          1232*             MOVE.B    STATR1(A5),D7           ;GET STATUS OF UART
                        1233* ;
                        1234* ; If Receive interrupt then see if should process character.
                        1235* ;
03BE 0807 0003          1236* DCIrcv      BTST      #S_RCVF, D7             ;TEST FOR RECEIVE BUFFER FULL
03C2 6702               1237*             BOFF.S    DCIxmit                 ;isn't, try Xmit buffer empty
03C4 6116               1238*             BSR.S     FRcvChar                ;yes, process receive character
                        1239* ;
                        1240* ; Not Receive, if Transmit interrupt then see if can send character
                        1241* ;       NOTE: THIS TESTS D7 WHICH ALLOWS US TO COME THRU HERE AFTER A READ CHECK DONE
                        1242*
03C6 0807 0004          1243* DCIxmit     BTST      #S_WRTE,D7              ;XMIT BUFFER EMPTY?
03CA 6704               1244*             BOFF.S    DCIexit                 ;NO, UNKNOWN INTERRUPT - EXIT
03CC 6100 0108          1245* DCIFX       BSR       FXMIT                   ;YES, PROCESS XMIT
03D0 4CDF 7FFF          1246* DCIexit     MOVEM.L   (SP)+,D0-A6             ;EXIT-RESTORE REGISTERS
03D4 4E73               1247*             RTE                               ;EXIT INTERRUPT
```

```
                        1249* ; PRcvChar - process received character
                        1250* ;      Entry : D7 = status register
                        1251* ;              D4 = unit number
                        1252* ;              A5 = UART Base address
                        1253* ;              A6 = port's data area address
                        1254* ;
03D6  162D  0001        1255* PRcvChar    MOVE.B    DATAREG(A5), D3         ;GET CHAR/CLEARS INTERRUPT
                        1256* ;;;;;;;;    BCLR      #BITD7,D3       ;don't so can send 8 bit characters
                        1257*
                        1258* ; check for any errors with receive
                        1259* ;
03DA  1007              1260*             MOVE.B    D7, D0                  ;get status register
03DC  0200  0007        1261*             ANDI.B    #S_ErrBits, D0          ;remove all but error bits
03E0  662E              1262*             BNE.S     PRCerror                ;have an error
                        1263*
                        1264* ; is this a control char and (protocols enabled) and (NOT Line type)
                        1265* ;   if yes then process com control char
                        1266* ;
03E2  082E  0000  0012  1267*             BTST      #PROT_P2, RF_PROF(A6)   ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
03E8  6712              1268*             BOFF.S    PRCnoctl                ; No protocol enabled, see if can put in bu
ffer
03EA  082E  0000  0013  1269*             BTST      #Line, RF_PROF+1(A6)    ; is it a Line protocol?
03F0  660A              1270*             BON.S     PRCnoctl                ; Yes
03F2  0C03  0020        1271*             CMPI.B    #' ', D3                ; is it a control character?
03F6  6404              1272*             BCC.S     PRCnoctl                ; No, not in range 0 - $1F
03F8  6126              1273*             BSR.S     PDCcontl                ; Yes, process a possible DC control char
03FA  6732              1274*             BEQ.S     PRCexit                 ; returns zero if processed a ctl char
                        1275*
                        1276* ; check to see if input disabled or buffer full
                        1277* ;
03FC  082E  0000  0033  1278* PRCnoctl    BTST      #FULL_R2, RB_FLG2+1(A6) ;is it Full?
0402  6614              1279*             BON.S     PRClstdt                ;Lost data error
0404  082E  0005  0031  1280*             BTST      #INPE_R1, RB_FLG1+1(A6) ;is input disabled?
040A  660C              1281*             BON.S     PRClstdt
                        1282*
                        1283* ; put char in buffer
                        1284* ;
040C  6000  005E        1285*             BRA       PutChrBf
                        1286*
                        1287* ; receive errors
                        1288* ;
0410  08EE  0001  0031  1289* PRCerror    BSET      #ERR_R1, RB_FLG1+1(A6)  ;UART error
0416  6006              1290*             BRA.S     PRCexit
0418  08EE  0001  0033  1291* PRClstdt    BSET      #LOST_R2, RB_FLG2+1(A6) ;Lost data error
041E  4E75              1292* PRCexit     RTS
```

```
                    1294* ;
                    1295* ; PDCcontl - check for Data Com control characters - ENQ, ETX, ACK, XON, and XOFF.
                    1296* ;
                    1297* ;       Entry : A6 = address of port's data area
                    1298* ;               D3 = character
                    1299* ;       Exit  : (NE) = char not one of the control characters
                    1300* ;               (EQ) = char was one of the control characters
                    1301* ;
0420  082E  0001  0013  1302* PDCcontl   BTST      #XONXOFF, HF_PROF+1(A6)  , is it XON/XOFF protocol?
0426  662E              1303*             BON.S     PDCLchkx                 ; yes, chk for those chars
                    1304*
                    1305* ; is either ENQ/ACK or ETX/ACK  both work the same way
                    1306* ;
0428  0C03  0006        1307*             CMPI.B    #ACK, D3                 ,is it an ACK?
042C  671C              1308*             BEQ.S     PDCLack                  ;yes, write is not busy now
042E  0C03  0005        1309*             CMPI.B    #ENQ, D3                 ;is it an ENQ?
0432  6706              1310*             BEQ.S     PDCLenq                  ;yes, see if read should go busy
0434  0C03  0003        1311*             CMPI.B    #ETX, D3                 ,is it an ETX?
0438  6630              1312*             BNE.S     PDCLexit                 ,no, not a control character
                    1313*
                    1314* , PROCESS an ENQ or ETX - send ACK if read not busy
                    1315* ;
043A  082E  0000  0031  1316* PDCLenq     BTST      #BUSY_R1, HB_FLG1+1(A6)  ,is read busy?
0440  6626              1317*             BON.S     PDCLdidit                ;yes, send ACK when clear Busy
0442  7006              1318*             MOVEQ     #ACK, D0                 ;no, then send ACK to other side
0444  6100  FF2C        1319* PDCLsend    BSR       PutCtl
0448  601E              1320*             BRA.S     PDCLdidit
                    1321*
                    1322* ; PROCESS an ACK and a XON - clear write busy
                    1323* ;
044A                1324* PDCLxon
044A  08AE  0000  0015  1325* PDCLack     BCLR      #BUSY_W1, WB_FLG1+1(A6)
0450  6100  FF06        1326*             BSR       STARTXMIT                ;start sending again
0454  6012              1327*             BRA.S     PDCLdidit
                    1328*
                    1329* ; Check for a XON or a XOFF
                    1330* ;
0456  0C03  0011        1331* PDCLchkx    CMPI.B    #XON, D3                 ,is it an XON?
045A  67EE              1332*             BEQ.S     PDCLxon                  ;yes, write is not busy now
045C  0C03  0013        1333*             CMPI.B    #XOFF, D3                ;is it an ENQ?
0460  6608              1334*             BNE.S     PDCLexit                 ,no, not a control character
                    1335*
                    1336* ; PROCESS a XOFF character - set write busy
                    1337* ;
0462  08EE  0000  0015  1338* PDCLxoff    BSET      #BUSY_W1, WB_FLG1+1(A6)
                    1339*
0466  4283              1340* PDCLdidit   CLR.L     D3                       ;show processed
0468  4E75              1341* PDCLexit    RTS
```

```
                        1343* ;PutChrBf --PUT A CHARACTER INTO THE READ BUFFER AND RETURN -ADJUST COUNTERS/POINTERS AS REQ
UIRED
                        1344* ;           COMING IN  D7- CONTAINS STATUS WORD  D3 CONTAINS CHARACTRER
                        1345* ;                      AS POINTS TO UART
                        1346* ;
                        1347* ; BUFFER HAS ENUF SPACE, JUST ADD CHARACTER
                        1348* ;
0466  206E  0034        1349* PutChrBf    MOVEA.L    RB_FILLP(A6), A0          , address where to put character
0470  10C3              1350*             MOVE.B     D3, (A0)+                 ; AUTO ADJUST POINTER
0472  2D48  0034        1351*             MOVE.L     A0, RB_FILLP(A6)          ; RESET THE FILL POINTER RB_FILLP
                        1352* ;
                        1353* ; Update buffer variables
                        1354* ;
0476  226E  003C        1355*             MOVE.L     RB_BADR(A6), A1           ;A1 = ADDRESS OF BUFFER BEGIN
047A  D2EE  0040        1356*             ADDA.W     RB_SIZE(A6), A1           ;A1 = ADDRESS OF END OF BUFFER
047E  B1C9              1357*             CMPA.L     A1, A0                    ;Is Rear pointing beyond buffer?
0480  6304              1358*             BLS.S      PCBnowrp                  ;No, don't do wrap around
                        1359* ;                                                 ;Yes, set front = addr 1st byte of buffer
0482  2D6E  003C  0034  1360*             MOVE.L     RB_BADR(A6), RB_FILLP(A6) ;Save the new Rear pointer in rb_fillp
                        1361* ;
0488  536E  0042        1362* PCBnowrp    SUBQ.W     #1, RB_FREE(A6)           ;SINCE WE put in a CHAR, 1 LESS FREE SPACE
                        1363*
                        1364* ; see if buffer is full  (RB_FREE is an unsigned word)
                        1365* ;
048C  6606              1366*             BNE.S      PCBnotfl                  ;not full, subtract sets or clears ZERO bit
048E  08EE  0000  0033  1367*             BSET       #FULL_B2, RB_FLG2+1(A6)   , BUFFER full
                        1368*
                        1369* ; buffer for sure is not empty
                        1370* ;
0494  08AE  0001  0033  1371* PCBnotfl    BCLR       #EMPT_B2, RB_FLG2+1(A6)   ;RESET EMPTY FLAG ANYHOO
                        1372*
                        1373* ; if protocols enabled and NOT Line type protocol then check buffer for hi water mark
                        1374* ;
049A  082E  0000  0012  1375*             BTST       #PROT_B2, BF_PROF(A6)     ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
04A0  670A              1376*             BOFF.S     PCBexit                   ; No protocol enabled, exit
04A2  082E  0000  0013  1377*             BTST       #Line, BF_PROF+1(A6)      , is it a Line protocol?
04A8  6602              1378*             BON.S      PCBexit                   , Yes, exit
04AA  6102              1379*             BSR.S      ChkRcvBusy                ; check for receive busy state
                        1380* ;
04AC  4E75              1381* PCBexit     RTS
```

```
                        1383* ,
                        1384* ; ChkRcvBusy - check for receive busy
                        1385*
                        1386* ; is size of buffer now at or above high water mark
                        1387* ;
04AE  322E  0040        1388* ChkRcvBusy  MOVE.W    RB_SIZE(A6), D1          ,BUFFER SIZE (ADDRESS OF)
04B2  926E  0042        1389*            SUB.W     RB_FREE(A6), D1          ,D1 = number of chars in buffer
04B6  B26E  004A        1390*            CMP.W     RB_HIWA(A6), D1          , at or above hi water mark?
04BA  6402              1391*            BCC.S     GoRcvBusy                , Yes, goto busy state
04BC  4E75              1392* CRBExit    RTS                                ,No, then exit
                        1393* ;
                        1394* ; GoRcvBusy - goto the Receive busy state
                        1395* ; assumes interrupts are turned off
                        1396* ,
04BE  08EE  0006 0031   1397* GoRcvBusy   BSET      #BUSY_R1, RB_FLG1+1(A6)  ,set busy state
04C4  668E              1398*            BNE.S     CRBSexit                 ; already busy so dont send char, exit
04C6  082E  0001 0013   1399*            BTST      #XONXOFF, SF_PROF+1(A6)  ;send byte to other side saying not busy
04CC  6706              1400*            BEQF.S    CRBSexit                 ,only if XON/XOFF protocol
04CE  7013              1401*            MOVEQ     #XOFF, D0                ,send XOFF
04D0  6100  FEA0        1402*            BSR       PutCtl                   ;put in control char buffer
04D4  4E75              1403* CRBSexit   RTS                                ;if ETX/ACK or ENQ/ACK nothing else to do
```

```
                        1405* ; PRIMIT - process transmission interrupt
                        1406* ,          Just send the next character if possible
                        1407* ,
                        1408* ,          ENTRY : (A5) = UART Base address
                        1409* ;                  (A6) = address of port's data area
                        1410* ;                  (D4) = unit number
                        1411* ;                  (D7) = status byte from UART
                        1412* ,
04D6  082E  0001  0057  1413* PRIMIT     BTST      #EMPT_CB, CB_FLAGS+1(A6)  ;control char available?
04DC  671E              1414*            BOFF.S    PRIgetctl                 ;yes, send it out next
                        1415*
                        1416* ; if NOT Busy or Buffer not empty send out next character
                        1417* ,
04DE  082E  0000  0015  1418*            BTST      #BUSY_W1, WB_FLG1+1(A6)   ;Busy?
04E4  660C              1419*            BON.S     PRIoff                    ;yes, turn off xmit int
04E6  082E  0001  0017  1420*            BTST      #EMPT_W2, WB_FLG2+1(A6)   ;buffer empty?
04EC  670A              1421*            BOFF.S    PRIsend                   ;NO, send next char
                        1422*
                        1423* ; check for an Alternate buffer available
                        1424* ,
04EE  6100  007C        1425*            BSR       wChkAltBf
                        1426*
                        1427* ; turn off xmit ints
                        1428* ;
04F2  6100  FE68        1429* PRIoff     BSR       STOPIMIT
04F6  6006              1430*            BRA.S     PRIexit
                        1431*
                        1432* , get next character in buffer and send out
                        1433* ;
04F8  6110              1434* PRIsend     BSR.S     SendNext
04FA  6002              1435*            BRA.S     PRIexit
                        1436*
                        1437* ; get next control character and send it out
                        1438* ;
04FC  6102              1439* PRIgetctl   BSR.S     SendCtl
                        1440* ;
04FE  4E75              1441* PRIexit     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  0001        1450*             MOVE.B    D3, DATAREG(A5)       , PUSH CHARACTER OUT
0508  4E75              1451*             RTS
```

```
                        1453* ; SendNext - Put next character in write buffer in UART transmit bufer.
                        1454* ;           ENTRY : (A5) = UART Base address
                        1455* ;                   (A6) = address of port's data area
                        1456* ;                   (D4) = unit number
                        1457* ;
050A  206E  001C        1458* SendNext    MOVE.L    WB_EMPTY(A6), A0        ;A0 =) EMPTYING POSITION OF RD BUFFER
050E  1B58  0001        1459*             MOVE.B    (A0)+, DATAREG(A5)      ; send out the next character
0512  2D48  001C        1460*             MOVE.L    A0, WB_EMPTY(A6)        ;Save the new Front pointer in rb_empty
                        1461*
                        1462* ; Update buffer variables
                        1463* ,
0516  224E  0020        1464*             MOVE.L    WB_BADR(A6), A1         ,A1 = ADDRESS OF BUFFER BEGIN
051A  D2EE  0024        1465*             ADDA.W    WB_SIZE(A6), A1         ;A1 = ADDRESS OF END OF BUFFER
051E  B1C9              1466*             CMPA.L    A1, A0                  ;Is Front pointing beyond buffer?
0520  6306              1467*             BLS.S     SNInowrp                ;No, don't do wrap around
                        1468* ;                                             ;Yes, set front = addr 1st byte of buffer
0522  2D6E  0020  001C  1469*             MOVE.L    WB_BADR(A6), WB_EMPTY(A6) ;Save the new Front pointer in wb_empty
                        1470* ;
0528  526E  0026        1471* SNInowrp    ADDQ.W    #1, WB_FREE(A6)         ;SINCE WE GOT CHAR, ONE MORE FREE SPACE
052C  08AE  0000  0017  1472*             BCLR      #FULL_W1, WB_FLG1+1(A6) ;always not full
                        1473*
                        1474* , see if buffer is empty
                        1475* ;
0532  322E  0026        1476*             MOVE.W    WB_FREE(A6), D1
0536  B26E  0024        1477*             CMP.W     WB_SIZE(A6),.D1         ; # OF FREE LOCATIONS - BUFFER SIZE
053A  650A              1478*             BCS.S     SNInotmt                ; not empty if free ( size
053C  6220              1479*             BHI.S     SNIsserr                ; size error *BUG if happens* (free ) size)
                        1480*
053E  08EE  0001  0017  1481*             BSET      #EMPT_W2, WB_FLG2+1(A6) ; BUFFER empty/turn off int next occurance i
n PRINIT
0544  6126              1482*             BSR.S     WChkAltBf               , see if should switch to an Alternate buffe
r
                        1483*
                        1484* ; if Protocols enabled and either ENQ/ACK or ETX/ACK then check
                        1485* ;      if should send an ENQ or ETX
                        1486* ,
0546  41EE  0012        1487* SNInotmt   LEA       BF_PROF(A6), A0
054A  0810  0000        1488*             BTST      #PROT_P2, (A0)+         ;protocol enabled?
054E  671A              1489*             BOFF.S    SNIexit                 ;no, exit
0550  0810  0002        1490*             BTST      #ENQACK, (A0)           ;ENQ/ACK protocol?
0554  6606              1491*             BON.S     SNIcnt                  ;yes, see if should send an ENQ
0556  0810  0007        1492*             BTST      #ETXACK, (A0)           ;ETX/ACK protocol?
055A  670E              1493*             BOFF.S    SNIexit                 ;no, exitt
055C  6038              1494* SNIcnt      BRA.S     CntChars                ;check if time to send ENQ or ETX
                        1495*
                        1496* ; Size error - set Error flag and split
                        1497* ;
055E  08EE  0001  0015  1498* SNIsserr    BSET      #ERR_W1, WB_FLG1+1(A6)  ;show size error
0564  08EE  0001  0017  1499*             BSET      #EMPT_W2, WB_FLG2+1(A6) ;  mark BUFFER empty
056A  4E75              1500* SNIexit     RTS
```

```
                    1502* ;
                    1503* ; WChkAltBf - check if alternate buffer switch on write buffer
                    1504* ;      Xmit input is automatically disabled when user calls unitstatus
                    1505* ;      to switch buffers.
                    1506* ;
                    1507* ;      Entry :  A6 = address of port's data area
                    1508* ;               interrupts disabled
                    1509* ;
056C 082E 0001 0017 1510* WChkAltBf   BTST       #EMPT_W1, WB_FLG1+1(A6)  ;is buffer empty?
0572 6720           1511*            BEQF.S      wCABexit                 ;no, can't switch
0574 45EE 0015      1512*            LEA         WB_FLG1+1(A6), A2        ;A2 = address of write buffer flags 1
0578 0892 0002      1513*            BCLR        #ALTBF_W1, (A2)          ;is an alternate buffer available?
057C 6716           1514*            BEQF.S      wCABexit                 ;no, nothing to switch
                    1515* ;
                    1516* ; Switch buffers by making the Alternate buffer the main buffer
                    1517* ,
057E 206E 0028      1518*            MOVE.L      WB_ABADR(A6), A0         ;get new buffer address
0582 302E 002C      1519*            MOVE.W      WB_ABSIZE(A6), D0        ;and length
0586 6100 0464      1520*            BSR         SetupWB                  ;switch buffer in table (doesnt use A2)
                    1521* ,
                    1522* ; if user is NOT controlling the Xmit input disable bit then enable
                    1523* ;
058A 0812 0007      1524*            BTST        #INPC_W1, (A2)           ;is user controlling input disable?
058E 6604           1525*            BNE.S       wCABexit                 ;user is controlling, exit
0590 0892 0005      1526*            BCLR        #INPE_W1, (A2)           ;no, enable input to buffer from user
                    1527* ;
0594 4E75           1528* wCABexit    RTS
```

```
                        1530* ;
                        1531* ; CntChars - see if sent enough characters to send out an ENQ or ETX
                        1532* ;
                        1533* ;       Entry :   A6 = address of port's data area
                        1534* ;                 (A5) = UART Base address
                        1535* ;                 (D4) = unit number
                        1536* ;                 Protocol is either ENQ/ACK or ETX/ACK
                        1537* ;
0596  526E  002E        1538* CntChars    ADDQ.W    #1, WB_BENQ(A6)          ;add 1 to char count between ctl chars
059A  302E  000E        1539*             MOVE.W    BF_BTWNEA(A6), D0        ;get max allowed between
059E  B06E  002E        1540*             CMP.W     WB_BENQ(A6), D0          ;did send max?
05A2  621A              1541*             BHI.S     CNTexit                  ;no, then exit
                        1542*
                        1543* ; set max chars between last ENQ or ETX, send another and go busy until receive ACK
                        1544* ,
05A4  7005              1545*             MOVEQ     #ENQ, D0                 ;assume send an ENQ
05A6  082E  0002  0013  1546*             BTST      #ENQACK, BF_PROF+1(A6)   ;ENQ/ACK protocol?
05AC  6602              1547*             BON.S     CNTenq                   ;yes
05AE  7003              1548*             MOVEQ     #ETX, D0                 ;is ETX/ACK protocol, send an ETX
05B0  6100  FDC0        1549* CNTenq      BSR       PutCtl                   ;put char in control char buffer
05B4  426E  002E        1550*             CLR.W     WB_BENQ(A6)              ;clear in between count
05B8  08EE  0000  0015  1551*             BSET      #BUSY_WI, WB_FLG1+1(A6)  ;go write busy
05BE  4E75              1552* CNTexit     RTS
```

```
                        1554* ;
                        1555* ; DCTLINT - Data Com Control interrupt service routine.
                        1556* ;
                        1557* ;           Makes check for both Ports.
                        1558* ;           Calls the routine at address saved in SETVECS routine during install.
                        1559* ;           Assumes it will clear the interrupt, toggle IOX.  The last routine in the
                        1560* ;           chain should be the OS Level 1 routine which does turn off the interrupt.
                        1561* ;
                        1562* ;           Ignores the interrupt if wasn't a DataCom Control interrupt,
                        1563* ;           therefore an Apple slot interrupt, or if NOT Line type
                        1564* ;           handshake method.
                        1565* ;
05C0  48E7  FFFE        1566* DCTLINT     MOVEM.L   D0-A6,-(SP)             ;SAVE REGISTERS
05C4  4DFA  0468+       1567*             LEA       Port0Data, A6           ;do for Port 0 first
05C8  383A  091E+       1568*             MOVE.W    UnitP0, D4              ;unit number of Port 0
05CC  6114              1569*             BSR.S     ChkLines
05CE  4DFA  04BA+       1570*             LEA       Port1Data, A6           ;do for Port 1 first
05D2  383A  0916+       1571*             MOVE.W    UnitP1, D4              ;unit number of Port 1
05D6  610A              1572*             BSR.S     ChkLines
                        1573*
                        1574* ; exit by restoring registers and then going to routine at saved address
                        1575* ;
05D8  4CDF  7FFF        1576*             MOVEM.L   (SP)+,D0-A6             ;EXIT-RESTORE REGISTERS
05DC  2F3A  090E+       1577*             MOVE.L    SaveLvl1, -(SP)         ;fake interrupt start
05E0  4E75              1578*             RTS
```

```
                        1580* ;
                        1581* ; ChkLines - see if change in lines for the port$specified by A6 and D4
                        1582* ;
                        1583* ;       Entry : D4 = unit number for the current port
                        1584* ;               A6 = address of data area for the current port
                        1585* ;
05E2  4286              1586* ChkLines    CLR.L     D6                      ;for disabling interrupts
05E4  6100  FB2A        1587*             BSR       GETBASE                 ;setup UART base reg (A5) *2/9/83*
                        1588*
                        1589* ;  See if any protocols at all and if so if any are line prots
                        1590* ;
05E8  082E  0000  0012  1591*             BTST      #PROT_F2, BF_PROF(A6)
05EE  6730              1592*             BOFF.S    CLNexit                 ;NO PROTOCOLS--GET OUT
                        1593* ;
                        1594* ; If (type of handshake () Line) then exit
                        1595* ;
05F0  082E  0000  0013  1596*             BTST      #LINE, BF_PROF+1(A6)
05F6  6728              1597*             BOFF.S    CLNexit                 ;NOT LINE HANDSHAKE, EXIT
                        1598*
                        1599* , Determine which Line is used as Busy line Port A
                        1600* ,
05F8  6128              1601*             BSR.S     FINDLIN                 ;NEEDS D4 = Unit number of current port
                        1602*                                               ;returns bit number to check in D5
                        1603* ,
                        1604* ;set or clear Busy depending on state of line and whether it's Busy inverted or not
                        1605* ;
05FA  162E  0015        1606*             MOVE.B    WB_FLG1+1(A6), D3       ;SAVE BUSY FLAG
05FE  6100  FA96        1607*             BSR       DISINTS                 ;DISABLE INTS
0602  08EE  0000  0015  1608*             BSET      #BUSY_W1, WB_FLG1+1(A6) ;ASSUME LINE IS BUSY = TRUE
0608  6134              1609*             BSR.S     TSTLINE                 ;TEST LINE & INVERTED FLAG (clobbers D1 & D
2)
060A  6610              1610*             BNE.S     CLNenbl                 ;IS BUSY
060C  08AE  0000  0015  1611*             BCLR      #BUSY_W1, WB_FLG1+1(A6) ;not busy
                        1612* ;
                        1613* ; if wasn't Busy before then start up transmission process
                        1614* ;
0612  0803  0000        1615*             BTST      #BUSY_W1,D3             ;TEST SAVED BUSY STATE
0616  6704              1616*             BOFF.S    CLNenbl                 ;WASN'T BUSY
0618  6100  FD3E        1617*             BSR       STRTXMIT                ;START XMIT IF BUFFER NOT EMPTY
                        1618* ;
                        1619* ; enable interrupts
                        1620* ;
061C  6100  FA9E        1621* CLNenbl     BSR       ENBINTS
                        1622* ;
0620  4E75              1623* CLNexit     RTS
```

```
                    1625* ;
                    1626* ; changed function to return value in D5 *1-12-83 kb*
                    1627* ; FINDLIN - Find which Line is used for Handshaking in Port A
                    1628* ;           ENTRY : (D4) = unit number
                    1629* ;                   Is a Line type protocol.
                    1630* ;           EXIT  : (D5) = Bit # in Port A specifying line used for Busy
                    1631* ,
0622  7A01          1632* FINDLIN     MOVEQ     #1,D5                   ;BIT NUMBER IN PORT A CORRESPONDING TO
0624  7203          1633*             MOVEQ     #CTSLIN,D1              ,FLAG BIT NUMBER
                    1634* ;
                    1635* ; Assumes that it will always find a line flag set
                    1636* ,
0626  032E  0013    1637* FLNLOOK     BTST      D1, BF_PROF+1(A6)       ,IS BIT SET? *1-12-83 kb*
062A  660A          1638*             BOM.S     FLNGOT                  ;YES, D3 PORT A BIT FOR DC 0
062C  5405          1639*             ADDQ.B    #2,D5
062E  5201          1640*             ADDQ.B    #1,D1                   ;TRY NEXT BIT FLAG
0630  0C01  0006    1641*             CMPI.B    #DCDLIN+1, D1           ,DID LAST FLAG
0634  66F0          1642*             BNE.S     FLNLOOK                 ;NO
                    1643* ,
                    1644* ; if (Port 1 is unit number) then bit# := bit# + 1 - DC 1 bits in Port A are next bit up
                    1645* ,
0636  B87A  0000+   1646* FLNGOT      CMP.W     UnitP#, D4              ,is it Port 0?
063A  6702          1647*             BEQ.S     FLNEXIT                 ;yes, then exit
063C  5205          1648*             ADDQ.B    #1,D5                   ,no, then  Port 1 and add 1 to bit number
063E  4E75          1649* FLNEXIT     RTS
                    1650* ,
                    1651* ; changed function to receive bit number parameter in D5 *1-12-83 kb*
                    1652* , TSTLINE - test Port A line used for Busy and the inverted flag to show if
                    1653* ,           Busy or NOT Busy.
                    1654* ,           ENTRY : (D5) = bit number in Port A of Line used by Busy
                    1655* ,           EXIT  : (NE) = Busy - D2 = $FF
                    1656* ,                   (EQ) = NOT Busy - D2 = $00
                    1657* ;
0640  0839  0003 0F7F 1658* TSTLINE     BTST      D5, MHIRA.L             ,Create Line Boolean
0644  56C1          1659*             SNE       D1
0646  082E  0006 0013 1660*             BTST      #INVBUSY, BF_PROF+1(A6) ,Create Inverted Boolean
064E  56C2          1661*             SNE       D2
0650  8302          1662*             EOR.B     D1,D2                   ,IF RESULT IS $FF THEN BUSY
0652  4E75          1663*             RTS
```

```
                        1665* ; COMCLR - UNITCLEAR
                        1666* ,          Initialize Buffers to empty.  Initialize Communications control
                        1667* ;          variables.  Initialize UART from Printer Control Table.
                        1668* ;
0654  6100  FA40        1669* COMCLR     BSR       DISINTS                 ,DISABLE INTERRUPTS
0658  6100  FA8A        1670*            BSR       SETUART                 ,INIT UART FROM CONSTANTS & TABLE
065C  6108              1671*            BSR.S     InitBufs                ;initialize read and write buffers to empty
065E  6158              1672*            BSR.S     ClrBusy                 ;clear busy flags
0660  6100  FA5A        1673*            BSR       ENBINTS                 ;ENABLE INTERRUPTS
0664  4E75              1674*            RTS
                        1675* ;
                        1676* ; InitBufs - initialize read, write and control buffers to empty
                        1677* ;          enable out and in bound on both buffers, remove all buffers
                        1678* ;      Exit : (D0) = old busy flag for read buffer
                        1679* ;             (D1) = old busy flag for write buffer
                        1680* ;
0666  2D6E  003C  0034  1681* InitBufs    MOVE.L    RB_BADR(A6), RB_FILLP(A6) ;initialize front and
066C  2D6E  003C  0038  1682*            MOVE.L    RB_BADR(A6), RB_EMPTY(A6) ; rear pointers
0672  3D6E  0040  0042  1683*            MOVE.W    RB_SIZE(A6), RB_FREE(A6)  ;show count as all free
0678  426E  0032        1684*            CLR.W     RB_FLG2(A6)               ; reset AutoLf, send LF, Full and Lost
067C  08EE  0001  0033  1685*            BSET      #EMPT_R2, RB_FLG2+1(A6)   ; BUFFER IS EMPTY
0682  302E  0030        1686*            MOVE.W    RB_FLG1(A6), D0           ;GET old busy flag
0686  426E  0030        1687*            CLR.W     RB_FLG1(A6)               ;reset all flags
                        1688*
068A  2D6E  0020  0018  1689*            MOVE.L    WB_BADR(A6), WB_FILLP(A6) ;initialize front and
0690  2D6E  0020  001C  1690*            MOVE.L    WB_BADR(A6), WB_EMPTY(A6) ; rear pointers
0696  3D6E  0024  0026  1691*            MOVE.W    WB_SIZE(A6), WB_FREE(A6)  ;show count as all free
069C  426E  0016        1692*            CLR.W     WB_FLG2(A6)               ; reset send LF, Full and Lost
06A0  08EE  0004  0017  1693*            BSET      #AULF_W2, WB_FLG2+1(A6)   ; DO AUTO LF and *kb 1/5/83*
06A6  08EE  0001  0017  1694*            BSET      #EMPT_W2, WB_FLG2+1(A6)   ; BUFFER IS EMPTY
06AC  322E  0014        1695*            MOVE.W    WB_FLG1(A6), D1           ;GET old busy flag
06B0  426E  0014        1696*            CLR.W     WB_FLG1(A6)               ;reset all flags
                        1697*
06B4  6000  FAEE        1698*            BRA       INITCTLB                  ,init ctl char buffer
                        1699* ,
                        1700* ; ClrBusy - if Read buffer was busy then send out NOT busy state
                        1701* ;          ignore write busy for now
                        1702* ;
                        1703* ;      Entry : (D0) = old busy flag for read buffer
                        1704* ;              (D1) = old busy flag for write buffer
                        1705* ;
06B8  0800  0000        1706* ClrBusy     BTST      #BUSY_R1, D0
06BC  6704              1707*            BEQ.S     CBSexit                   ;may have to check if have
06BE  6100  FBB2        1708*            BSR       GoUnBusy                  ;protocols and not line type
06C2  4E75              1709* CBSexit     RTS
```

```
                        1711* ;
                        1712* ; COMBSY - UNITBUSY
                        1713* ;          PASCAL BOOLEAN TRUE RETURNED IN D0 IF THERE ARE ANY CHARACTERS IN READ BUFFER
                        1714* ;
          06C8          1715* COMBSY
          06C8 082E 0001 0033 1716*          BTST      #EMPT_RI, RB_FLG3+1(A4)
          06CE 57C0           1717*          SEQ       D0                      ;IF BIT NOT SET THEN = 0; CHARACTRERS EXIST
D0 =111111
          06CC 0200 0001      1718*          ANDI.B    #TRUE,D0                ;CONVERT FROM BOOLEAN TO PASCAL BOOLEAN-
          06D0 4E75           1719*          RTS
                        1720* ;
                        1721* ; COMUNMT - UNITUNMOUNT
                        1722* ;          Turnoff interrupt capabilities of COMM  driver.
                        1723* ;          Restore vectors.
                        1724* ;
          06D2 6100 F9C2      1725* COMUNMT     BSR       DISIMTS                 ;DISABLE INTERRUPTS
                        1726* ;
          06D6 6100 FA38      1727*          BSR       GETBASE                 ;GET UART BASE
          06DA 1B7C 0002 0005 1728*          MOVE.B    #TURNOFF,CMDREG1(A5)    ;TURNOFF UART
                        1729* ;
                        1730* ; have vectors point to a RTE instruction
                        1731* ;
          06E0 41F8 0070      1732*          LEA       VEC4.W, A0              ;assume it is Port 0
          06E4 43F8 0068      1733*          LEA       VEC1.W, A1
          06E8 45FA 001A+     1734*          LEA       THERTE, A2              ;address of the RTE instruction
          06EC B87A 07FA+     1735*          CMP.W     UnitP0, D4              ;is it Port 0?
          06F0 6702           1736*          BEQ.S     CUM:sP0                 ;yes, change level 4
          06F2 C348           1737*          EXG       A0, A1                  ;no, change level 1
          06F4 208A           1738* CUM:sP0     MOVE.L    A2, (A0)                ;set vector to point at RTE
                        1739* ;
                        1740* ; if both vectors point at RTE then set level 1 to saved address
                        1741* ;
          06F6 B388           1742*          CMPM.L    (A0)+, (A1)+            ;must do post inc.
          06F8 6606           1743*          BNE.S     CUMdiff                 ;different so not both RTE
          06FA 21FA 07F0+ 0064 1744*          MOVE.L    SaveLvl1, VEC1.W        ;restore from saved area
                        1745* ;
                        1746* ; Restore Interrupts
                        1747* ;
          0700 6000 F9BA      1748* CUMdiff     BRA       ENBINTS
                        1749*
                        1750* ; THE RTE INSTRUCTION
                        1751* ;
          0704 4E73           1752* THERTE      RTE
```

```
                        1754* ;
                        1755* ; COMST - UNITSTATUS
                        1756* ; call the Table change or buffer free Functions
                        1757* ;
0706  0C42  0014        1758* COMST     CMPI.W    #TBLSTATE,D2          ;VALID FUNCTION CODE
070A  6210              1759*           BHI.S     CSTERR                ;NO
070C  3013              1760*           MOVE.W    (A3),D0               ;GET PARAMETER
070E  43FA  0010+       1761*           LEA       CSTTBL,A1             ;TURN THE FUNCTION CODE INTO
0712  E34A              1762*           LSL.W     #1,D2                 ;AN INDEX TO THE FUNCTION
0714  3431  2000        1763*           MOVE.W    0(A1,D2.W),D2
0718  4EF1  2000        1764*           JMP       0(A1,D2.W)            ;DO FUNCTION
                        1765* ;
                        1766* ; Invalid Function Code Error
                        1767* ;
071C  7E38              1768* CSTERR     MOVEQ     #INVFNC,D7
071E  4E75              1769*           RTS
                        1770* ;
                        1771* ; THE CON DRIVER STATUS JUMP TABLE
                        1772* ;
                        1773* ; functions compatible with old printer driver
                        1774*
0720  002E              1775* CSTTBL    DATA.W    STWBUF-CSTTBL         ;WRITE BUFFER FREE SPACE
0722  0040              1776*           DATA.W    STBAUD-CSTTBL         ;SET BAUD RATE
0724  0050              1777*           DATA.W    STPRITY-CSTTBL        ;SET PARITY
0726  0038              1778*           DATA.W    STRBUF-CSTTBL         ;READ BUFFER FREE SPACE
0728  0060              1779*           DATA.W    STWRDSZ-CSTTBL        ;SET WORD SIZE
072A  007E              1780*           DATA.W    STHNDSK-CSTTBL        ;SET HANDSHAKE METHOD
072C  0144              1781*           DATA.W    STBFSTS-CSTTBL        ;TELL BUFFER CONTROL STATUS
                        1782*
                        1783* ; new functions
                        1784*
072E  00D2              1785*           DATA.W    STRDSTS-CSTTBL        ;TELL READ STATUS
0730  0122              1786*           DATA.W    STWTSTS-CSTTBL        ;TELL WRITE STATUS
0732  00B6              1787*           DATA.W    STRDHI-CSTTBL         ;SET READ BUFFER HI WATER MARK
0734  00BE              1788*           DATA.W    STRDLO-CSTTBL         ;SET READ BUFFER LOW WATER MARK
0736  0180              1789*           DATA.W    STOUTRD-CSTTBL        ;TOGGLE OUTBOUND READ
0738  01C0              1790*           DATA.W    STINRD-CSTTBL         ;TOGGLE INBOUND READ
073A  01D6              1791*           DATA.W    STOUTWT-CSTTBL        ;TOGGLE OUTBOUND WRITE
073C  01F2              1792*           DATA.W    STINWT-CSTTBL         ;TOGGLE INBOUND WRITE
073E  0232              1793*           DATA.W    BWBCHR-CSTTBL         ;TELL #CHARS IN WRITE BUFFER
0740  023E              1794*           DATA.W    BRBCHR-CSTTBL         ;TELL #CHARS IN READ BUFFER
0742  00C4              1795*           DATA.W    STATOLF-CSTTBL        ;TOGGLE auto LineFeed flag
0744  00CC              1796*           DATA.W    STBENQ-CSTTBL         ;SET number of chars between ENQ's
0746  024A              1797*           DATA.W    STRDALTBF-CSTTBL      ;SET Read Alternate buffer
0748  0292              1798*           DATA.W    STWTALTBF-CSTTBL      ;SET Write Alternate bufferen ENQ's
```

```
074A 6000 F94A      1800* STCallDis  BRA       DISINTS                 ;DISABLE INTERRUPTS
                    1801* ;
                    1802* ; STWBUF - Return to the user the Free space in the write buffer
                    1803* ;
074E 61FA           1804* STWBUF     BSR.S     STCallDis               ;DISABLE INTERRUPTS
0750 36AE 0026      1805*            MOVE.W    WB_FREE(A6), (A3)       ;WRITE BUFFER FREE SPACE
0754 6000 F966      1806* STCallEnb  BRA       ENBINTS                 ;ENABLE INTERRUPTS
                    1807* ;
                    1808* ; STRBUF - Return to the user the Free space in the READ buffer
                    1809* ;
0758 61F0           1810* STRBUF     BSR.S     STCallDis               ;DISABLE INTERRUPTS
075A 36AE 0042      1811*            MOVE.W    RB_FREE(A6), (A3)       ;WRITE BUFFER FREE SPACE
075E 60F4           1812*            BRA.S     STCallEnb               ;ENABLE INTERRUPTS
                    1813* ;
                    1814* ; STBAUD - Set the Baud Rate
                    1815* ;
0760 0C40 0004      1816* STBAUD     CMPI.W    #MAXBAUD,D0             ;IS IT A VALID PARAMETER
0764 6234           1817*            BHI.S     SETERR                  ;NO
                    1818* ;
0766 41EE 000B      1819*            LEA       BF_RDBD(A6), A0         ;WHERE TO PUT VALUE
076A 43FA 02AC+     1820*            LEA       BAUDCNV,A1              ;CONVERSION ARRAY
076E 601E           1821*            BRA.S     SAVPARM                 ;SAVE CONVERTED PARAMETER
                    1822* ,
                    1823* ; STPRITY - Set the Parity
                    1824* ;
0770 0C40 0004      1825* STPRITY    CMPI.W    #MAXPRTY,D0             ;IS IT A VALID PARAMETER
0774 6224           1826*            BHI.S     SETERR                  ;NO
                    1827* ,
0776 41EE 000C      1828*            LEA       BF_PART(A6), A0         ;WHERE TO PUT VALUE
077A 43FA 02A3+     1829*            LEA       PRTYCNV,A1              ;CONVERSION ARRAY
077E 600E           1830*            BRA.S     SAVPARM                 ;SAVE CONVERTED PARAMETER
                    1831* ;
                    1832* , STWRDSZ - Set the word size to transmit (7 or 8)
                    1833* ,
0780 0C40 0001      1834* STWRDSZ    CMPI.W    #MAXWRDS,D0             ;IS IT A VALID PARAMETER
0784 6214           1835*            BHI.S     SETERR                  ;NO
                    1836* ;
0786 41EE 000D      1837*            LEA       BF_WRDS(A6), A0         ;WHERE TO PUT VALUE
078A 1080           1838*            MOVE.B    D0,(A0)                 ;PUT IN WORD SIZE VALUE
078C 6004           1839*            BRA.S     RSTUART                 ;RESET UART FROM TABLE
                    1840* ;
                    1841* ; common code to STBAUDR, STPRITY, STWRDSZ, STDTACOM, & STHNDSK
                    1842* ;
078E 10B1 0000      1843* SAVPARM    MOVE.B    0(A1,D0.W),(A0)         ;SAVE CONVERTED PARAMETER
                    1844* ;
0792 6186           1845* RSTUART    BSR.S     STCallDis               ;DISABLE INTERRUPTS
0794 6100 F94E      1846* RSTUART1   BSR       SETUART                 ;SETUP UART FROM TABLE
0798 60BA           1847*            BRA.S     STCallEnb               ;ENABLE INTERRUPTS
                    1848* ;
                    1849* ; Invalid Parameter error
                    1850* ;
079A 7E36           1851* SETERR     MOVEQ     #INVPRM,D7
079C 4E75           1852*            RTS
```

```
                        1854* ;
                        1855* ; STHNDSK - Set Handshake type.  Convert parameter into the flags and put these
                        1856* ;           flag values into the Printer Control Table.  Don't need to reset
                        1857* ;           UART.
                        1858* ;
07?E                    1859* STHNDSK
07?E  0C40  0009        1860*           CMPI.W    #MAXHNDS,D0             ;IS IT A VALID PARAMETER
07A2  62F6              1861*           BHI.S     SETERR                  ;NO
                        1862* ;
07A4  43FA  027E+       1863*           LEA       HNDSCNV,A1              ;CONVERSION ARRAY
07A8  1D71  0000  0013  1864*           MOVE.B    0(A1,D0.W), BF_PROF+1(A6) ;move new flags into flag byte
                        1865*
                        1866* ; see if user disabled all protocols
                        1867* ;
07AE  08EE  0000  0012  1868*           BSET      #PROT_P2, BF_PROF(A6)   ;assume have a protocol
07B4  4A2E  0013        1869*           TST.B     BF_PROF+1(A6)           ;IF zero then no protocols
07B8  6608              1870*           BNE.S     SHDchkEA                ;see if ETX/ACK or ENQ/ACK
07BA  08AE  0000  0012  1871*           BCLR      #PROT_P2, BF_PROF(A6)   ;show no protocol
07C0  6014              1872*           BRA.S     SHDexit
                        1873*
                        1874* ;
                        1875* ;
07C2  082E  0007  0013  1876* SHDchkEA  BTST      #ETXACK, BF_PROF+1(A6)  ;is it ETX/ACK?
07C8  6608              1877*           BON.S     SHDzero                 ;yes, zero char count
07CA  082E  0002  0013  1878*           BTST      #ENQACK, BF_PROF+1(A6)  ;is it ENQ/ACK?
07D0  6704              1879*           BOFF.S    SHDexit                 ;no, exit
07D2  426E  003E        1880* SHDzero   CLR.W     WB_SENQ(A6)             ;clr cnt of chars between ENQ's or ETX's
                        1881* ;
07D6  4E75              1882* SHDexit   RTS
```

```
                        1884* ;
                        1885* ;STRDHI  -SET THE READ BUFFER HIGH WATER MARK
                        1886* ;
07D8  3D53  004A        1887* STRDHI    MOVE.W    (A3), RB_HIWA(A6)
07DC  4E75              1888*           RTS
                        1889* ;
                        1890* ;STRDLO  -SET THE READ BUFFER LOW WATER MARK
                        1891* ;
07DE  3D53  004C        1892* STRDLO    MOVE.W    (A3), RB_LOWA(A6)
07E2  4E75              1893*           RTS
                        1894* ;
                        1895* ; STATLF - toggle the Auto LineFeed flag
                        1896* ;
07E4  086E  0004 0017   1897* STATOLF   BCHG      #AULF_V2, WB_FLG2+1(A6)  ;flip the bit
07EA  4E75              1898*           RTS
                        1899* ;
                        1900* ; STRENQ - set the number of chars between ENQ's or ETX's
                        1901* ;
07EC  3D53  000E        1902* STRENQ    MOVE.W    (A3), RF_RTWMEA(A6)
07F0  4E75              1903*           RTS
```

```
                    1905* ; STRDSTS - GET THE READ BUFFER STATUS
                    1906* ;       ParameterBlock = record
                    1907* ;                          BufferSize     : integer;
                    1908* ;                          FreeSpace      : integer;
                    1909* ;                          HiWater        : integer;
                    1910* ;                          LowWater       : integer;
                    1911* ;                          InputDisabled  : byte;    (true = 1, false = 0)
                    1912* ;                          OutputDisabled : byte;    (true = 1, false = 0)
                    1913* ;                          LostData       : byte;    (true = 1, false = 0)
                    1914* ;                          AltBufferAvail : byte;    (true = 1, false = 0)
                    1915* ;                          AltBufferAddr  : pointer; (0 if AltBufferAvail false)
                    1916* ;                          AltBufferSize  : integer; (0 if AltBufferAvail false)
                    1917* ;                        end;
                    1918* ;
07F2  36EE  0040    1919* STRDSTS     MOVE.W    RB_SIZE(A6), (A3)+      ;get buffer size
07F6  36EE  0042    1920*             MOVE.W    RB_FREE(A6), (A3)+      ;get free space byte count
07FA  36EE  004A    1921*             MOVE.W    RB_HIWA(A6), (A3)+      ;get hi water byte count
07FE  36EE  004C    1922*             MOVE.W    RB_LOWA(A6), (A3)+      ;get low water byte count
                    1923*
                    1924* ; get the flags and make byte Pascal booleans
                    1925* ;
0802  082E  0005  0031  1926*             BTST      #INPE_B1, RB_FLG1+1(A6) ;is PORT to BUFFER disabled?
0808  612E                1927*             BSR.S     MAKEBOOL
080A  082E  0004  0031  1928*             BTST      #OUTE_B1, RB_FLG1+1(A6) ;is BUFFER to USER disabled?
0810  6126                1929*             BSR.S     MAKEBOOL
0812  08AE  0002  0033  1930*             BCLR      #LOST_B2, RB_FLG2+1(A6) ;has any data been lost?
0818  611E                1931*             BSR.S     MAKEBOOL
                    1932*
                    1933* ; If have an Alt buffer then return it's ADDRESS AND SIZE
                    1934* ;
081A  082E  0002  0031  1935*             BTST      #ALTBF_B1, RB_FLG1+1(A6)
0820  670E                1936*             BOFF.S    RDSTnone
0822  16FC  0001  1937*             MOVE.B    #1, (A3)+               ;set Alt buffer boolean
0826  36EE  0044  1938*             MOVE.L    RB_ABADR(A6), (A3)+     ;get Alternate buffer Address
082A  36AE  0048  1939*             MOVE.W    RB_ASIZE(A6), (A3)      ;get Alternate buffer size
082E  6006                1940*             BRA.S     RDSTexit
                    1941*
0830  421B                1942* RDSTnone    CLR.B     (A3)+                   ;no Alternate buffer available
0832  4298                1943*             CLR.L     (A3)+                   ;so NIL pointer for address
0834  4253                1944*             CLR.W     (A3)                    ;and zero bytes size
                    1945*
0836  4E75                1946* RDSTexit    RTS
                    1947* ;
                    1948* ; MAKEBOOL - make Pascal boolean from zero flag
                    1949* ;
0838  56C0                1950* MAKEBOOL    SNE       D0              ;D0.B = $FF if zero flag clear
083A  0200  0001  1951*             ANDI.B    #TRUE, D0       ;turn to Pascal boolean (1 = true)
083E  16C0                1952*             MOVE.B    D0, (A3)+       ;save in parameter block
0840  4E75                1953*             RTS
```

```
                        1955* ; STWTSTS - GET THE WRITE BUFFER STATUS
                        1956* ;       ParameterBlock = record
                        1957* ;                          BufferSize     : integer;
                        1958* ;                          FreeSpace      : integer;
                        1959* ; FROM BUFFER CNTRL TBL -) CharsBtwnENQs  : integer;
                        1960* ;                          InputDisabled  : byte;    (true = 1, false = 0)
                        1961* ;                          OutputDisabled : byte;    (true = 1, false = 0)
                        1962* ;                          AutoLineFeed   : byte;    (true = 1, false = 0)
                        1963* ;                          AltBufferAvail : byte;    (true = 1, false = 0)
                        1964* ;                          AltBufferAddr  : pointer, (0 if AltBufferAvail false)
                        1965* ;                          AltBufferSize  : integer; (0 if AltBufferAvail false)
                        1966* ;                        end;
                        1967* ;
0842  36EE  0014        1968* STWTSTS     MOVE.W    WB_SIZE(A6), (A3)+      ;get buffer size
0846  36EE  0026        1969*             MOVE.W    WB_FREE(A6), (A3)+      ;get free space byte count
084A  36EE  000E        1970*             MOVE.W    BF_BTWNEA(A6), (A3)+    ,get maximum number of chars between ENQ's o
r ETX's
                        1971*
                        1972* ; get the flags and make byte Pascal booleans
                        1973* ;
084E  082E  0005  0015  1974*             BTST      #INPE_W1, WB_FLG1+1(A6) ;is USER to BUFFER disabled?
0854  61E2              1975*             BSR.S     MAKEBOOL
0856  082E  0004  0015  1976*             BTST      #OUTE_W1, WB_FLG1+1(A6) ;is BUFFER to PORT disabled?
085C  61DA              1977*             BSR.S     MAKEBOOL
085E  082E  0004  0017  1978*             BTST      #AULF_W2, WB_FLG2+1(A6) ;is Auto LineFeed mode on?.
0864  61D2              1979*             BSR.S     MAKEBOOL
                        1980*
                        1981* ; IF have an Alt buffer then return it's ADDRESS AND SIZE
                        1982* ;
0866  082E  0002  0015  1983*             BTST      #ALTBF_W1, WB_FLG1+1(A6)
086C  670E              1984*             BOFF.S    WTSTnone
086E  16FC  0001        1985*             MOVE.B    #1, (A3)+               ;set Alt buffer boolean
0872  16EE  0028        1986*             MOVE.L    WB_ABADR(A6), (A3)+     ;get Alternate buffer Address
0876  36AE  002C        1987*             MOVE.W    WB_ASIZE(A6), (A3)      ;get Alternate buffer size
087A  6006              1988*             BRA.S     WTSTexit
                        1989*
087C  421B              1990* WTSTnone    CLR.B     (A3)+                   ;no Alternate buffer available
087E  429B              1991*             CLR.L     (A3)+                   ;so NIL pointer for address
0880  4253              1992*             CLR.W     (A3)                    ;and zero bytes size
                        1993*
0882  4E75              1994* WTSTexit    RTS
```

```
                        1996* ; STBFSTS - Return to the user in the parameter block the state of the Buffer Control Table.
                        1997* ,       ParameterBlock = record
                        1998* ;                          BaudRate  : integer; ;(range = 0..6)
                        1999* ;                          Parity    : integer; ;(range = 0..4)
                        2000* ;                          DataCom   : integer; ;(range = 0..1)
                        2001* ;                          WordSize  : integer; ;(range = 0..1)
                        2002* ;                          HandShake : integer; ;(range = 0..9)
                        2003* ;                        end;
                        2004* ;
0884  4281              2005* STBFSTS     CLR.L     D1                      ;MAKE SURE NO GARBAGE IN REGISTER
                        2006* ;
                        2007* ; GET BAUD RATE
                        2008* ;
0886  303C  0006        2009*             MOVE.W    #MAXBAUD,D0             ;MAX BAUD RATE PARAMETER VALUE
088A  122E  000B        2010*             MOVE.B    BF_RDBD(A6), D1         ;CURRENT TABLE VALUE
088E  41FA  0188+       2011*             LEA       BAUDCNV,A0              ;CONVERT TO INTEGER RANGE
0892  612C              2012*             BSR.S     GETVAL
                        2013* ;
                        2014* ; GET PARITY
                        2015* ;
0894  303C  0004        2016*             MOVE.W    #MAXPRTY,D0             ;MAX PARITY PARAMETER VALUE
0898  122E  000C        2017*             MOVE.B    BF_PART(A6), D1         ;CURRENT TABLE VALUE
089C  41FA  0181+       2018*             LEA       PRTYCNV,A0              ;CONVERT TO INTEGER RANGE
08A0  611E              2019*             BSR.S     GETVAL
                        2020* ;
                        2021* ; GET DATACOM - BASED ON D4 and the SAVED UNIT NUMBER
                        2022* ;
08A2  4281              2023*             CLR.L     D1                      ;assume is Port 0
08A4  B87A  0642+       2024*             CMP.W     UnitP0, D4              ;is Port 0?
08A8  6702              2025*             BEQ.S     SBFSisP0                ;yes
08AA  7201              2026*             MOVEQ     #1, D1                  ;no, show as Port 1
08AC  36C1              2027* SBFSisP0    MOVE.W    D1, (A3)+               ;save parameter
                        2028* ;
                        2029* ; GET WORD SIZE
                        2030* ;
08AE  122E  000D        2031*             MOVE.B    BF_WRDS(A6), D1
08B2  36C1              2032*             MOVE.W    D1, (A3)+
                        2033* ;
                        2034* ; GET HANDSHAKE
                        2035* ;
08B4  303C  0009        2036*             MOVE.W    #MAXHNDS,D0             ;MAX HANDSHAKE PARAMETER VALUE
08B8  122E  0013        2037*             MOVE.B    BF_PROF+1(A6),D1        ;CURRENT TABLE VALUE
08BC  41FA  0164+       2038*             LEA       HNDSCNV,A0              ;CONVERT TO INTEGER RANGE
                        2039* ;
                        2040* ; GET PARAMETER VALUE AN PUT IN PARAMETER BLOCK
                        2041* ;
08C0  B230  0000        2042* GETVAL      CMP.B     0(A0,D0.W), D1          ;SEE WHICH CONVERSION VALUE = CURRENT VALUE
08C4  57C8  FFFA        2043*             DBEQ      D0, GETVAL              ;THE INDEX OF ONE = IS THE PARAMETER VALUE
TO
08C8  36C0              2044*             MOVE.W    D0, (A3)+               ;RETURN TO USER IN PARAMETER BLOCK
08CA  4E75              2045*             RTS
```

```
08CC  6000  F7C8        2047* DoDisInt    BRA       DISINTS                 ;DISABLE INTERRUPTS
                        2048* ;
                        2049* ; STOUTRD -- TOGGLE OUTBOUND RECEIVE DISABLE  (BUFFER TO USER)
                        2050* ;
08D0  61FA              2051* STOUTRD     BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08D2  086E  0006  0031  2052*             BCHG      #OUTC_R1, RB_FLG1+1(A6)
08D8  086E  0004  0031  2053*             BCHG      #OUTE_R1, RB_FLG1+1(A6)
08DE  6012              2054*             BRA.S     DoEnbInt                ;enable interrupts
                        2055* ;
                        2056* ; STINRD -- TOGGLE INBOUND RECEIVE DISABLE  (PORT TO BUFFER)
                        2057* ;
08E0  61EA              2058* STINRD      BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08E2  41EE  0031        2059*             LEA       RB_FLG1+1(A6), A0       ;address of flags
08E6  0850  0007        2060*             BCHG      #INPC_R1, (A0)          ;user currently controlling?
08EA  6604              2061*             BON.S     INRDenb                 ;yes, then enable
08EC  6148              2062*             BSR.S     DisRcvIn                ;no, disable input
08EE  6002              2063*             BRA.S     DoEnbInt
08F0  6136              2064* INRDenb     BSR.S     EnbRcvIn
                        2065*
08F2  4000  F7C8        2066* DoEnbInt    BRA       ENBINTS                 ;ENABLE INTERRUPTS
                        2067* ;
                        2068* ; STOUTWT -- TOGGLE OUTBOUND TRANSMIT DISABLE  (BUFFER TO PORT)
                        2069* ;
08F6  61D4              2070* STOUTWT     BSR.S     DoDisInt                ;DISABLE INTERRUPTS
08F8  086E  0004  0015  2071*             BCHG      #OUTC_W1, WB_FLG1+1(A6) ;toggle user controlling
08FE  086E  0004  0015  2072*             BCHG      #OUTE_W1, WB_FLG1+1(A6) ;and enable/disable flag
0904  6606              2073*             BON.S     OTWToff                 ;now disabled, turn off xmit int
0906  6100  FA50        2074*             BSR       STRTXMIT                ;enable xmit int
090A  60E6              2075*             BRA.S     DoEnbInt
090C  6100  FA4E        2076* OTWToff     BSR       STOPXMIT                ;disable xmit ints
0910  60E0              2077*             BRA.S     DoEnbInt                ;enable interrupts
                        2078* ;
                        2079* ; STINWT -- TOGGLE INBOUND TRANSMIT DISABLE  (USER TO BUFFER)
                        2080* ;
0912  61B8              2081* STINWT      BSR.S     DoDisInt                ;DISABLE INTERRUPTS
0914  41EE  0015        2082*             LEA       WB_FLG1+1(A6), A0       ;address of flags
0918  0850  0007        2083*             BCHG      #INPC_W1, (A0)          ;toggle user controlling
091C  0810  0002        2084*             BTST      #ALTBF_W1, (A0)         ;if got an alt buffer
0920  66D0              2085*             BON.S     DoEnbInt                ;then already set, let it enable
0922  0850  0005        2086*             BCHG      #INPE_W1, (A0)          ;else toggle it
0926  60CA              2087*             BRA.S     DoEnbInt                ;enable interrupts
```

```
                        2089* ;
                        2090* ; EnbRcvIn - Enable input receive
                        2091* ;       Entry : A6 = address of ports data area
                        2092* ;               A0 = address of read buffer flag 1 low byte
                        2093* ;               interrupts disabled
                        2094* ;
0928  0810  0002        2095* EnbRcvIn    BTST      #ALTBF_R1, (A0)          ;alternate buffer available?
092C  6604              2096*             BON.S     ERIchkprt                ;yes then let switch enable RCV
092E  0890  0005        2097*             BCLR      #INPE_R1, (A0)           ;NO, enable input
                        2098*
                        2099* ; see if should tell other side not BUSY
                        2100* ;
0932  6000  F910        2101* ERIchkprt   BRA       ChkProto
                        2102* ;
                        2103* ; DisRcvIn
                        2104* ;       Entry : A6 = address of ports data area
                        2105* ;               A0 = address of read buffer flag 1 low byte
                        2106* ;               interrupts disabled
                        2107* ;
0936  08D0  0005        2108* DisRcvIn    BSET      #INPE_R1, (A0)           ;disable input
093A  6614              2109*             BON.S     DRIexit                  ;if was off then don't go busy again
                        2110*
                        2111* ; if protocols enabled and NOT Line type protocol then go busy
                        2112* ;
093C  082E  0000  0012  2113*             BTST      #PROT_P2, BF_PROF(A6)    ; SEE IF ANY PROTOCOLS AT ALL--CHECK HI BYT
E
0942  670C              2114*             BOFF.S    DRIexit                  ; No protocol enabled, exit
0944  082E  0000  0013  2115*             BTST      #Line, BF_PROF+1(A6)     ; is it a Line protocol?
094A  6604              2116*             BON.S     DRIexit                  ; Yes, exit
094C  6100  FB70        2117*             BSR       GoRcvBusy                ; go busy
                        2118*
0950  4E75              2119* DRIexit     RTS
```

```
                        2121*  ;
                        2122*  ; BWBCHR -  GET Number of characters  IN the WRITE BUFFER
                        2123*  ,
0952  302E  0024          2124*  BWBCHR      MOVE.W     WB_SIZE(A6), D0          ;SIZE IN D1
0956  906E  0026          2125*              SUB.W      WB_FREE(A6), D0          ; SIZE - FREE = Number of CHARS
095A  3680                2126*              MOVE.W     D0, (A3)                 ; return to user amount
095C  4E75                2127*              RTS
                        2128*  ;
                        2129*  ; BRBCHR -  GET Number of characters  IN the READ BUFFER
                        2130*  ,
095E  302E  0040          2131*  BRBCHR      MOVE.W     RB_SIZE(A6), D0          ;SIZE IN D1
0962  906E  0042          2132*              SUB.W      RB_FREE(A6), D0          ; SIZE - FREE = Number of CHARS
0966  3680                2133*              MOVE.W     D0, (A3)                 , return to user amount
0968  4E75                2134*              RTS
```

```
                        2136* ;
                        2137* ; STRDALTBF - set Alternate Buffer for Read
                        2138* ;
096A  6100  F72A        2139* STRDALTBF   BSR       DISINTS                 ; disable interrupts
096E  6138              2140*             BSR.S     GetAltBuf               ; get address and size user passed
0970  660A              2141*             BNE.S     RDABsz                  ; addr ok, check size
0972  41EE  005C        2142*             LEA       RDBUF(A6), A0           ; they were zero so use default
0976  303C  0100        2143*             MOVE.W    #RBFLEN, D0             ; read buffer
097A  6008              2144*             BRA.S     RDABok
                        2145*
097C  4A40              2146* RDABsz       TST.W     D0                      ; check size, is it negative?
097E  6A04              2147*             BPL.S     RDABok                  ; no
0980  7E36              2148* STBFerr      MOVEQ     #INVPRM, D7             ; yes, invalid parameter
0982  6020              2149*             BRA.S     RDABexit
                        2150*
                        2151* ; got buffer address and length
                        2152* ;
0984  082E  0001  0033  2153* RDABok       BTST      #EMPT_R2, RB_FLG2+1(A6) ; current buffer empty?
098A  6616              2154*             BON.S     RDABswtch               ; yes, then use user's buffer
                        2155*
                        2156* ; buffer isn't empty so wait till empty to switch
                        2157* ;
098C  2D48  0044        2158*             MOVE.L    A0, RB_ABADR(A6)        ;save address in alt buffer adr
0990  3D40  0048        2159*             MOVE.W    D0, RB_ASIZE(A6)        ;and length in alt buffer size
0994  41EE  0031        2160*             LEA       RB_FLG1+1(A6), A0       ;DisRcvIn needs A0 -> flag byte
0998  08D0  0002        2161*             BSET      #ALTBF_R1, (A0)         ;alt buffer available true
099C  6100  FF98        2162*             BSR       DisRcvIn                ;disable input and see if should go busy
09A0  6002              2163*             BRA.S     RDABexit                ; exit
                        2164*
                        2165* ; EMPTY SO MAKE NEW the current buffer
                        2166* ;
09A2  6150              2167* RDABswtch    BSR.S     SetupRB
                        2168*
09A4  6000  F716        2169* RDABexit     BRA       ENBINTS
                        2170* ;
                        2171* ; GetNewBuf - from user's parameter block get the Alt buffer
                        2172* ;             address and size.
                        2173* ;      EXIT   D0 = alt buffer length
                        2174* ;             D1 = 0
                        2175* ;             A0 = alt buffer address
                        2176* ;             (EQ) = use default
                        2177* ;             (NE) = use A0 and D0
                        2178* ;
09A8  205B              2179* GetAltBuf    MOVEA.L   (A3)+, A0
09AA  301B              2180*             MOVE.W    (A3)+, D0
09AC  4281              2181*             CLR.L     D1
09AE  B288              2182*             CMP.L     A0, D1
09B0  4E75              2183* GABFexit     RTS
```

```
                        2185* ; STWTALTBF - set Alternate Buffer for Write
                        2186* ;
09B2  6100  F6E2        2187* STWTALTBF   BSR       DISINTS                 ; disable interrupts
09B6  61F0              2188*             BSR.S     GetAltBuf               ;get user's buffer address and size
09B8  660A              2189*             BNE.S     WTABsz                  , addr good chk size
09BA  41EE  015C        2190*             LEA       WRTBUF(A6), A0          , they were zero so use default
09BE  303C  0100        2191*             MOVE.W    #WBFLEN, D0             , write buffer
09C2  6004              2192*             BRA.S     WTABok
                        2193*
09C4  4A40              2194* WTABsz      TST.W     D0                      ; check size, is it negative?
09C6  6BB8              2195*             BMI.S     STBFerr                 , no
                        2196*
                        2197* , got buffer address and length
                        2198* ,
09C8  082E  0001  0017  2199* WTABok      BTST      #EMPT_W2, WB_FLG2+1(A6) , current buffer empty?
09CE  6616              2200*             BNE.S     WTABswtch               , yes, then use user's buffer
                        2201*
                        2202* , buffer isn't empty so wait till empty to switch
                        2203* ;
09D0  2D48  0028        2204*             MOVE.L    A0, WB_ABADR(A6)        ,save address in alt buffer adr
09D4  3D40  002C        2205*             MOVE.W    D0, WB_ASIZE(A6)        ;and length in alt buffer size
09D8  08EE  0005  0015  2206*             BSET      #INPE_W1, WB_FLG1+1(A6) ,disable input
09DE  08EE  0002  0015  2207*             BSET      #ALTBF_W1, WB_FLG1+1(A6) ,alt buffer available true
09E4  6002              2208*             BRA.S     WTABexit                ; exit
                        2209*
                        2210* , EMPTY SO MAKE NEW the current buffer
                        2211* ,
09E6  6104              2212* WTABswtch   BSR.S     SetupWB
                        2213*
09E8  6000  F6D2        2214* WTABexit    BRA       ENBINTS
                        2215* ,
                        2216* ; SetupRB - put the alternate buffer info in the Read Buffer Control Table
                        2217* , SetupWB - put the alternate buffer info in the Write Buffer Control Table
                        2218* ;      Entry : D0 = alternate buffer size
                        2219* ,              A0 = alternate buffer address
                        2220* ;
09EC  43EE  0018        2221* SetupWB     LEA       WB_FILLP(A6), A1
09F0  99CC              2222*             SUBA.L    A4,A4
09F2  6008              2223*             BRA.S     STUPgo
                        2224*
09F4  43EE  0034        2225* SetupRB     LEA       RB_FILLP(A6), A1
09F8  49EE  005C        2226*             LEA       RDBUF(A6), A4
                        2227*
                        2228* , move the buffer address into the Front, Rear, and buffer pointers
                        2229* ,
09FC  22C8              2230* STUPgo      MOVE.L    A0, (A1)+               ;set the fill (Front) pointer
09FE  22C8              2231*             MOVE.L    A0, (A1)+               ,set the empty (Rear) pointer
0A00  22C8              2232*             MOVE.L    A0, (A1)+               ;set the buffer pointer
                        2233*
                        2234* ; move the size into the buffer size and free space counter
                        2235* ;
0A02  32C0              2236*             MOVE.W    D0, (A1)+               ;set the size
0A04  32C0              2237*             MOVE.W    D0, (A1)+               ;set the free space available
                        2238*
```

```
                        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
```

```
                    1247* ;
                    1248* ; constant data area
                    1249* ;
                    1250* ; Conversion arrays for Set functions of Unitstatus
                    1251* ;
0A18  06 07 08 0A 0C 0E 1252* BAUDCNV     DATA.B     6,7,8,$A,$C,$E,$F          ;BAUD RATE
0A1E  0F
                    1253* , 6=300,7=600,8=1200,A=2400,C=4800,E=9600,F=19200
                    1254* ;
0A1F  00 01 03 05 07    1255* PRTYCNV     DATA.B     0,1,3,5,7                  ;PARITY
                    1256* ; 0=DISABLED,1=ODD,3=EVEN,5=MARK XMIT/NO RCV,7=SPACE XMIT/NO RCV
                    1257* ;
0A24  49                1258* HNDSCNV     DATA.B     $49                        ;LINE/CTS/INV
0A25  09                1259*              DATA.B     $09                        ;LINE/CTS/NOT INV
0A26  51                1260*              DATA.B     $51                        ;LINE/DSR/INV
0A27  11                1261*              DATA.B     $11                        ;LINE/DSR/NOT INV
0A28  61                1262*              DATA.B     $61                        ;LINE/DCD/INV
0A29  21                1263*              DATA.B     $21                        ;LINE/DCD/NOT INV
0A2A  02                1264*              DATA.B     $02                        ;XON/XOFF
0A2B  04                1265*              DATA.B     $04                        ;ENQ/ACK
0A2C  80                1266*              DATA.B     $80                        ;ETX/ACK
0A2D  00                1267*              DATA.B     $00                        ,NONE OF THE ABOVE PROTOCOLS
                    1268* ;*****************************************************************************
```

```
                    2270* ,
                    2271* ; Variable data area
                    2272* ,
                    2273* , Port 0 data area
                    2274* ;
0A2E                2275* Port0Data
                    2276*
                    2277* , DEFAULT BUFFER  Control Table  - MUST HAVE SAME FIELD FORMAT AS BUFFER CONTROL TABLE
                    2278* ,
     00000000       2279* DEFBWRT     EQU       %-Port0Data
0A2E 0E             2280*             DATA.B    $0E                    ;WRITE BAUD RATE-9600
     00000001       2281* DEFBRD      EQU       %-Port0Data
0A2F 0E             2282*             DATA.B    $0E                    ;READ BAUD RATE-9600
     00000002       2283* DEFPART     EQU       %-Port0Data
0A30 00             2284*             DATA.B    $00                    ;PARITY-DISABLED
     00000003       2285* DEFWRDS     EQU       %-Port0Data
0A31 00             2286*             DATA.B    $00                    ;WORD SIZE = 8 BITS (1=7 BITS)
     00000004       2287* DEFBTWNEA   EQU       %-Port0Data            ;NUMBER OF CHARS BETWEEN
0A32 0050           2288*             DATA.W    80                     ;        ENQ's or ETX's
     00000006       2289* DEFINTRN    EQU       %-Port0Data
0A34 0000           2290*             DATA.W    $0000                  ;INTERNAL FLAG--all off
     00000008       2291* DEFPROT     EQU       %-Port0Data
0A36 0902           2292*             DATA.W    $0902                  ;PROTOCOL FLAG--Enabled - XON/XOFF
     0000000A       2293* DEFend      EQU       %-Port0Data
     00000005       2294* DEFBCTLN    EQU       (DEFend-DEFBWRT)/2     ;number of words in both tables
                    2295*
                    2296* , BUFFER CONTROL TABLE
                    2297* ;
     0000000A       2298* BFRCTL      EQU       %-Port0Data            ;Index to Buffer Control Table
     0000000A       2299* BF_WRBD     EQU       %-Port0Data            ;Index to WRITE BAUD RATE
0A38 00             2300*             DATA.B    0                      ;
     0000000B       2301* BF_RDBD     EQU       %-Port0Data            ;Index to READ BAUD RATE
0A39 00             2302*             DATA.B    0                      ;
     0000000C       2303* BF_PART     EQU       %-Port0Data            ;Index to PARITY
0A3A 00             2304*             DATA.B    0                      ;
     0000000D       2305* BF_WRDS     EQU       %-Port0Data            ;Index to WORD SIZE
0A3B 00             2306*             DATA.B    0                      ;
     0000000E       2307* BF_BTWNEA   EQU       %-Port0Data            ;Index to NUMBER OF CHARS BETWEEN
0A3C 0000           2308*             DATA.W    0                      ;        ENQ's or ETX's
     00000010       2309* BF_INTL     EQU       %-Port0Data            ;Index to INTERNAL FLAGS
0A3E 0000           2310*             DATA.W    0                      ;
     00000012       2311* BF_PROF     EQU       %-Port0Data            ;Index to PROTOCOL FLAGS-HANDSHAKE TYPE
0A40 0000           2312*             DATA.W    0                      ;
                    2313*
                    2314* ;           WRITE BUFFER CONTROL TABLE
                    2315* ;
     00000014       2316* WRTCTL      EQU       %-Port0Data            ;Index to WRITE BUFFER CONTROL TABLE
     00000014       2317* WB_FLG1     EQU       %-Port0Data            ;Index to FLAG WORD 1
0A42 0000           2318*             DATA.W    0
     00000016       2319* WB_FLG2     EQU       %-Port0Data            ;Index to FLAG WORD 2
0A44 0000           2320*             DATA.W    0
     00000018       2321* WB_FILLP    EQU       %-Port0Data            ;Index to BUFFER FILL POINTER  rear
0A46 00000000       2322*             DATA.L    0
     0000001C       2323* 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
0A51  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
0A61  00000000          2346*               DATA.L    0
      00000038          2347* RB_EMPTY      EQU       %-Port0Data             ,Index to BUFFER EMPTY POINTER front
0A64  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       %-Port0Data
      00000100          2383* RBFLEN      EQU       RBFend-RDBUF              ,READ BUFFER LENGTH
                        2384* ,
                        2385* ,  Write Buffer - 256 bytes
                        2386* ,
      0000015C          2387* WRTBUF      EQU       %-Port0Data               ;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       %-Port0Data
      00000100          2393* WBFLEN      EQU       WBFend-WRTBUF           ;WRITE BUFFER LENGTH
                        2394* ;
      0000025C          2395* pdlen       EQU       %-Port0Data             ;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
```

```
                    2440* ,
                    2441* ;Common area
                    2442* ;
                    2443* ; Flags
                    2444* ;
0EE6  0000          2445* CMNFLGS     DATA.W     0
                    2446*
                    2447* ; Unit Numbers for the ports
                    2448* ;
0EE8  0000          2449* UnitP0      DATA.W     0
0EEA  0000          2450* UnitP1      DATA.W     0
                    2451*
                    2452* ; Save of DataCom Ctrl interrupt vector
                    2453* ;
0EEC  00000000      2454* SaveLvl1    DATA.L     0
                    2455*
      00000000+     2456*             END        COMDRV
```

ACK	00000006	CKBUFERR	0002C6+	COMREX	0001F4+	D2_BENQ	00000012	DEFINTRM	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	000094+
BF_INTL	00000010	CLRD3D2	000000F3	CR	0000000D	D2_OUTRD	0000000B	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	00000040	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	00000040	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	00000006	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	00038F67	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	000664+	CURSON	00000002	DEFBTWNE	00000004	FULL_CB	00000000
CHKRCVBU	0004AE+	COMISP0	000040+	D2_ATLF	00000011	DEFBWRT	00000000	FULL_P2	00000003
CINBUFCT	000672+	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+
GBSISF0    000120+  IOEORDSB 0000003D   ORA      00030F63   RDABOK     000984+  SNDLF_W2 00000003
GCLEXIT    00039A+  IOEOWDSB 0000003F   OTWTOFF    00090C+  RDABSWTC   0009A2+  SNICNT     00055C+
GETALTBU   0009A8+  IOEPADER 00000044   OUTC_R1  00000006   RDABSZ     00097C+  SNIEXIT    00056A+
GETBASE    000110+  IOEPRMLN 00000037   OUTC_W1  00000006   RDBUF    0000005C   SNINOTMT   000546+
GETCTL     000384+  IOERSZER 00000042   OUTE_R1  00000004   RDCTL    00000030   SNINOWRP   000528+
GETVAL     0008C0+  IOETBLFL 00000033   OUTE_W1  00000004   RDSTEXIT   000836+  SNISZERR   00055E+
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    0002BA+  STATEMSK 00002000
HILOMSK  000000F0   IOEWNDBE 00000021   PDCLCHKI   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+  SBFSISF0   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    0007E+
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   STOPIMIT   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   STRTIMIT   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+
IOEXYBTE 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+ SITGETB    00035E+ UPCNOTCR   000342+  WB_ASIZE 0000002C  WRHOMEOF 0000000C
SVDELENT 00000090  SYSBYTES 00000186  UPCNOTFL   000326+  WB_BADR  00000020  WRHOMEPT 00000004
SVDSF    00000038  SYSKYBDF 00000184  UPCNOWRP   00031A+  WB_BENQ  0000002E  WRLENGTH 00000030
SVDSP4   0000004C  SYSWIN   00000005  UPRMSK   0000A000  WB_EMPTY 0000001C  WRLNGTHI 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    0000001B  WCABEXIT   000594+ WRTBUF   00000150
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 00000000  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  000300F0  UTTYP    00000017  WRCMD    00000002  XMITDIS  00000008
SVUBUSY  0000000C  UCCNOTMT   00022E+ VEC1     00000064  WRCURADR 00000008  XMITENB  00000004
SVUCLEAR 00000008  UCCNOWRP    000216+ VEC2     00000068  WRCURSX  00000016  XOFF     00000013
SVUINSTL 00000098  UCETCHR     0001F6+ VEC4     00000070  WRCURSY  00000018  XON      00000011
SVUISP0    0000E0+ UNDSCR   00000001  VERT     00000000  WRFILL1  00000025  XONXOFF  00000001
SVUREAD  00000004  UNITP0      000EE0+ 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.

```
    1. { PRTWND.TEXT ------------------------------------------------------------}
    2. {
    3. {        PRTWND -- Print Current Window
    4. {
    5. {        (c) Copyright 1982 Corvus Systems, Inc.
    6. {                           San Jose, California
    7. {
    8. {        All Rights Reserved
    9. {
   10. {        v 1.0  07-01-82  KML  Original program for MX100 printer
   11. {        v 1.1  10-01-82  LEF  Added IDS460 printer support
   12. {
   13. {-----------------------------------------------------------------------}
   14.
   15. PROGRAM prtwnd,
   16.
   17. USES {$U /CCUTIL/CCLIB} CCdefn, CCwndIO,
   18.
   19. CONST esc = 27,
   20.
   21. TYPE  prtrid = (NONE, IDS, MX100),
   22.
   23. VAR   prtype: prtrid;
   24.       prtr:   integer;
   25.       disp:   integer;
   26.       i,argn. integer;
   27.       curWnd: pWndRcd;
   28.       pDev:   pString80,
   29.
   30. { CCLIB external definitions     }
   31.
   32. FUNCTION  pOScurWnd: pWndRcd;    { get kybd record pointer } EXTERNAL,
   33. FUNCTION  pOSdevNam (n: integer): pString80;                 EXTERNAL,
   34. FUNCTION  OSdispDv: integer;     { get display unit nmbr    } EXTERNAL,
   35.
   36. PROCEDURE Rbytes (x,y,count. integer; pBuff: pBytes);
   37.       const RDBYTES = 7;
   38.       var wbuf. record
   39.                 bytecount: integer,
   40.                 buffptr. pBytes;
   41.                 end,
   42.       begin
   43.       if y < 0 then begin
   44.           pBuff^[0] := 0; exit (Rbytes), end,
   45.       write ('\1Bo', chr(x div 256), chr(x mod 256),
   46.                      chr(y div 256), chr(y mod 256), chr(2)),
   47.       wbuf.bytecount := count;
   48.       wbuf.buffptr   := pBuff;
   49.       unitstatus (disp,wbuf,RDBYTES);
   50.       end;
   51.
   52. {$P}
```

```
53. PROCEDURE spit (b: byte);
54.         begin unitwrite (6, b, 1, 0, 1); end;
55.
56. PROCEDURE doit;
57.         var i,j,x,y: integer; b: byte;
58.             cell: array [0..6] of byte;
59.
60.         FUNCTION bit (i,j: integer): integer;
61.             var b: integer;
62.             begin
63.             b := cell[i], bit := 0;
64.             if b < 0 then b := b+256;
65.             case j of
66.                 0: if odd (b div 128) then bit := 1;
67.                 1: if odd (b div  64) then bit := 1;
68.                 2: if odd (b div  32) then bit := 1;
69.                 3: if odd (b div  16) then bit := 1;
70.                 4: if odd (b div   8) then bit := 1;
71.                 5: if odd (b div   4) then bit := 1;
72.                 6: if odd (b div   2) then bit := 1;
73.                 7: if odd (b)         then bit := 1;
74.               end;
75.             end;
76.
77.         begin
78.         write ('\1Bb'); { CURSOR OFF }
79.         with curWnd^ do begin
80.
81. { SET LINE SPACING TO 8 DOTS }
82.
83.             if prtype = MX100 then begin
84.                 spit (esc), spit (ord('A')); spit (8); end;
85.             if prtype = IDS then begin
86.                 spit (3); end;
87.
88. { PRINT LEFT WINDOW BORDER }
89.
90.             if prtype = MX100 then begin
91.                 spit (esc); spit (ord('K'));
92.                 spit ((lngthy+2) mod 256);
93.                 spit ((lngthy+2) div 256);
94.                 for y := 0 to lngthy+1 do spit (1);
95.                 end;
96.
97. {$P}
```

```
 98. { PRINT WINDOW }
 99.
100.            if prtype = MX100 then
101.                for x := 0 to lngthx div 8 do begin
102.                    spit (13); spit (10);
103.                    spit (esc); spit (ord('K'));
104.                    spit ((lngthy+2) mod 256);
105.                    spit ((lngthy+2) div 256);
106.                    spit (-1); { BOTTOM WINDOW BORDER }
107.                    for y := 0 to lngthy-1 do begin
108.                        Rbytes (x*8, y, 1, @b);
109.                        spit (b);
110.                        end;
111.                    spit (-1); { TOP WINDOW BORDER }
112.                    end;
113.            if prtype = IDS then begin
114.                y := lngthy-1;
115.                repeat
116.                    for x := 0 to lngthx div 8 do begin
117.                        for i := 0 to 6 do
118.                            Rbytes (x*8, y-i, 1, @cell[i]);
119.                        for j := 0 to 7 do begin
120.                            b := bit (6,j) * 64 +
121.                                 bit (5,j) * 32 +
122.                                 bit (4,j) * 16 +
123.                                 bit (3,j) *  8 +
124.                                 bit (2,j) *  4 +
125.                                 bit (1,j) *  2 +
126.                                 bit (0,j);
127.                            if b = 3 then spit (3);
128.                            spit (b);
129.                            end;
130.                        end;
131.                    spit (3); spit (14);
132.                    y := y-7;
133.                    until y < 0;
134.                end;
135.
136. { PRINT RIGHT WINDOW BORDER }
137.
138.            if prtype = MX100 then begin
139.                spit (13); spit (10);
140.                spit (esc); spit (ord('K'));
141.                spit ((lngthy+2) mod 256);
142.                spit ((lngthy+2) div 256);
143.                for y := 0 to lngthy+1 do spit (-128);
144.                end;
145.
146. {$P}
```

```
147.  { NORMALIZE PRINTER }
148.
149.            if prtype = MX100 then begin
150.                spit (13); spit (10);
151.                spit (eso); spit (ord('2')); spit (12);
152.                end;
153.            if prtype = IDS then begin
154.                spit (3); spit (2); end;
155.            end;
156.        write ('\1Bo'); { CURSOR ON }
157.        end;
158.
159.
160.        begin
161.        CCwndIOinit;
162.        prtr := 6;
163.        disp := OSdispDv;
164.        ourWnd := pOSourWnd;
165.        pDev := pOSdevNam (prtr);
166.        if pDev^ <> 'PRINTER' then begin
167.            writeln ('Printer driver not loaded ....',chr(7)),
168.            exit (prtwnd);
169.            end;
170.        prtype := MX100;
171.        if argc <> 0 then begin
172.            prtype := NONE;
173.            for argn := 1 to argc do begin
174.                for i := 1 to length(argv[argn]^) do
175.                    if argv[argn]^[i] in ['a'..'z'] then
176.                       argv[argn]^[i] := chr(ord(argv[argn]^[i])
177.                                             -ord('a')+ord('A'));
178.                if argv[argn]^ = 'IDS'   then prtype := IDS;
179.                if argv[argn]^ = 'MX100' then prtype := MX100;
180.                end;
181.            end;
182.        if prtype <> NONE
183.            then doit
184.            else writeln ('Invalid printer type specified ....',chr(7));
185.        end.
186.
187.
188.
```

NOTE

THE FOLLOWING PAGES CONTAIN THE CORVUS CONCEPT KEYBOARD TRANSLATION TABLES.

Table of Contents
KEYBOARD TRANSLATION TABLES

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.

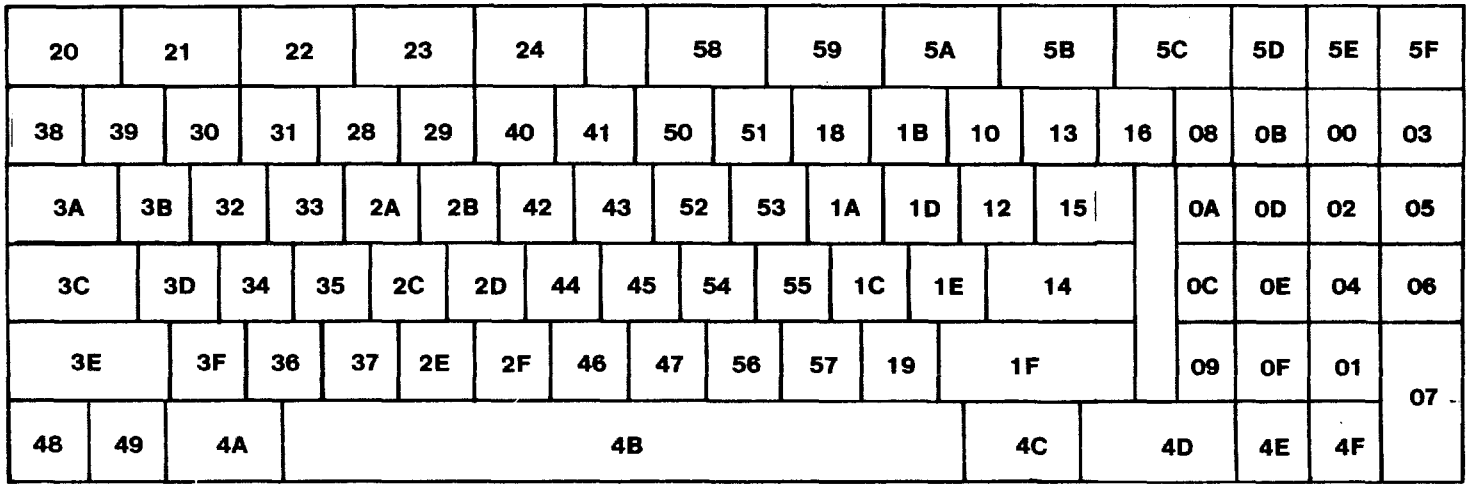
2.0 The Keyboard and Keycodes

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.



Keycode Map (release code)

Figure 1

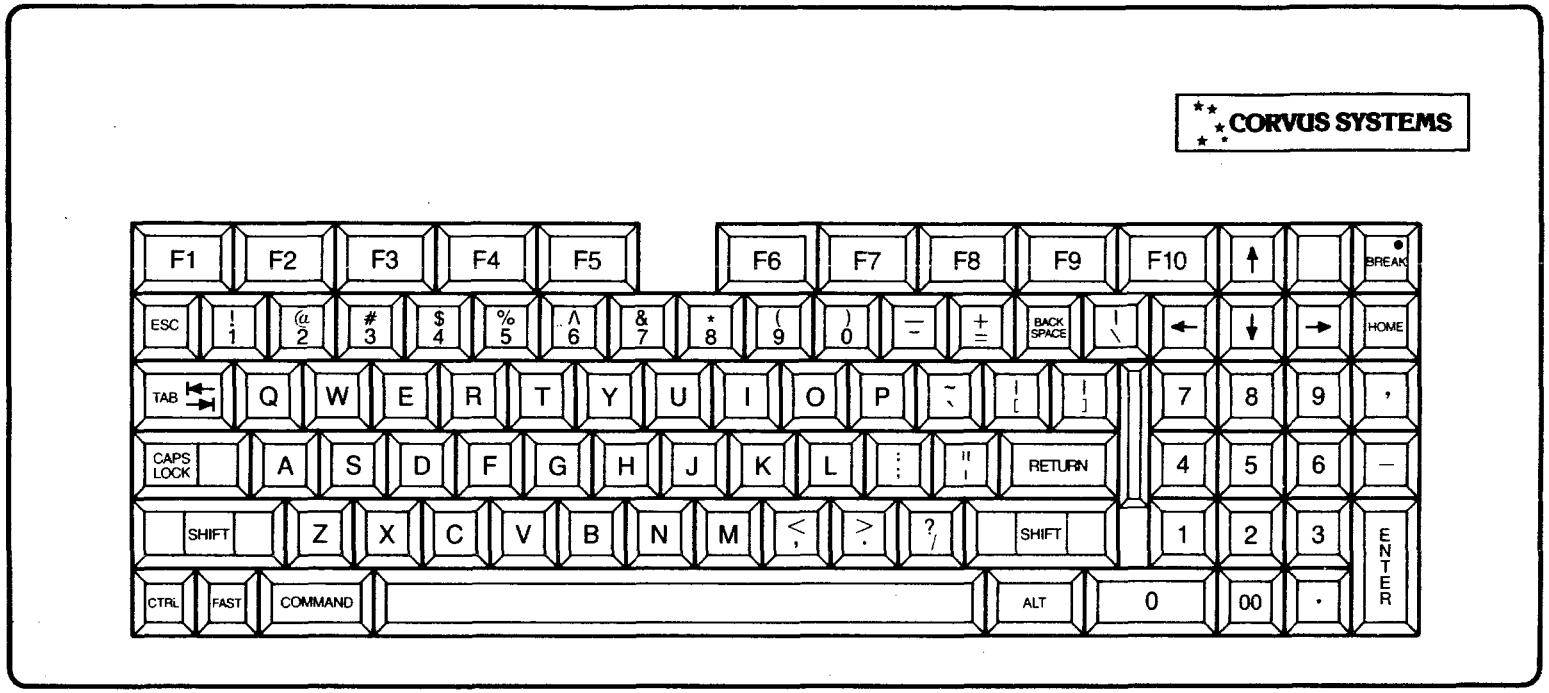

CORVUS SYSTEMS
F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
BREAK
ESC
BACK SPACE
HOME
TAB
CAPS LOCK
RETURN
SHIFT
SHIFT
CTRL
FAST
COMMAND
ALT
ENTER

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

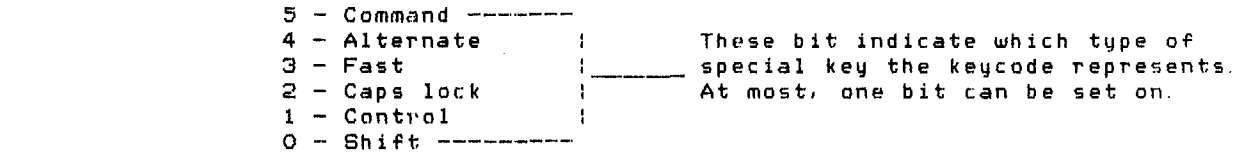
3.5 CAPS LOCK & QUALIFIER FLAG TABLE (CQTABLE)

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.



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
D2 = Caps lock
D1 = Control
D0 = Shift

This bit says which type of Qualifier key the Keycode represents.

```
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)
GUL = 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   -   ;   `   '  GUL
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  GUL  a  GUL  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  GUL GUL GUL  sp GUL  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 GUL GUL
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 GUL
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 nc 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  G  GUL  A  GUL  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  GUL GUL GUL sp  GUL  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 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  GUL  a  GUL  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  GUL GUL GUL sp  GUL  O  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 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.  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 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.  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   GUL
  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   GUL A   GUL 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   GUL GUL GUL sp  GUL 0   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)
;    GUL = 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   GUL
  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   GUL a   GUL 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   GUL GUL GUL sp  GUL 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 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
```